Machine Learning

Coursera - Stanford University

Andrew Ng

Summarised By:

Sanjay Singh

san.singhsanjay@gmail.com

Introduction

- Machine Learning is the science of getting computers to act without being explicitly programmed.
- You use ML dozens of times a day:
  - When you do a Google search
  - When Facebook tags you in a picture
  - Your spam filter
  - Etc.
- Machine Learning:
  - Grew out of work in AI: we wanted our machines to be as much intelligent as human. To do this, we realised the only way is ML.
  - New capability for computers: To make our computers better than earlier.
- Some examples of applications of ML:
  - Database mining: Large database from growth of automation / web.
    - Web click data
    - Medical Records
    - Biology – like genome data set
    - Engineering – almost every field
  - Applications cannot programmed by hand:
    - Autonomous helicopter
    - Handwritten recognition
    - Natural Language Processing (NLP)
    - Computer Vision
  - Self customization programs: Amazon and Netflix product recommendations.
  - Understanding human learning (brain, real AI).

Introduction

- There is no universally accepted definition of Machine Learning (ML).
- Following definition is an old one given by Arthur Samuel in 1959:

  "Machine Learning is the field of study that gives computers the ability to learn with out being explicitly programmed."
- The most recent definition of Machine Learning (ML) was given by Tom Mithcell in 1998:

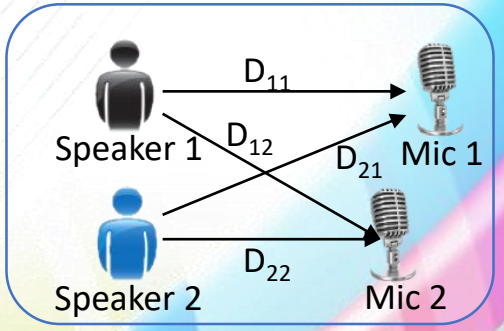
  "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."
- Mainly, there are two types of Machine Learning algorithms:
  1. Supervised Learning:

     Right answers are provided with each instance of data set for the learning of algorithm.
  2. Unsupervised Learning:

     Basically, it is a clustering technique. This is used to find out different unknown types or classes.
- Other ML algorithms are:
  1. Reinforcement Learning
  2. Recommender Systems

Introduction

- Cocktail Party problem can be solved by using Unsupervised Learning problem.
- Following is the Cocktail Party problem:
  - There are n number of speakers speaking simultaneously (suppose n = 2).
  - There are n number of mics (suppose n = 2).
  - These mics are at different distance from each speaker.
  - Each mic is comparatively closer to one speaker and farther from others.
  - Each of these mics will record the composed voice / signal of every speaker.
  - Problem is to separate out the voice (or signal) of each speaker.



- Above problem is one the most complicated problem in the field of Computer Science and Signal Processing. However, it can be solved with the help of Unsupervised Learning Algorithms.
- Following is a one line unsupervised solution for Cocktail Party problem:

*[W, s, v] = svd((repmat(sum(x.*x, 1), size(x, 1), 1). *x)*x');*

[Source: Sam Roweis, Yair Weiss & Eero Simoncelli]

Introduction

- We will use GNU Octave to implement these ML algorithms due to following reasons:
  - Octave is free and an open source alternative of MATLAB.
  - Octave has a good collection of computation and numerical libraries.
  - All the complicated mathematical functions are available in Octave, hence it would be quite fast and easy to implement them in Octave.
  - Doing the similar thing in any other language, like Python, C / C++, Java, etc., will take a long time and efforts.

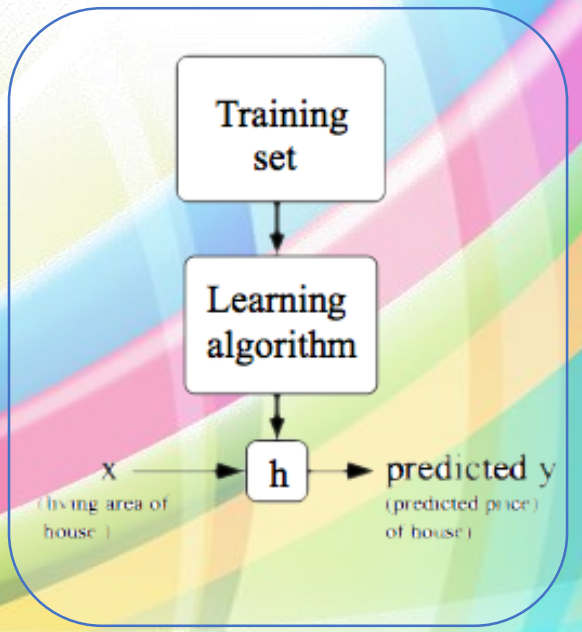
- Many people in Silicon Valley use Octave to prototype their Machine Learning algorithms as it is easy and quick. Only after getting a desired result / output, then start implementing it in the required programming language, like C / C++, Java, Python, etc.

Model and Cost Function

Notations

- Following are some notations used in the further lectures:

Notation	Explanation
m	Number of training examples or instances
x	Input variable or feature
y	Output or target variable or feature
(x, y)	One training instance or example (any)
(x^i, y^i)	i^{th} training instance or example
h	Hypothesis: the output function of a learning algorithm mapping input and output



Model and Cost Function

- The hypothesis function of machine learning model looks something like this:

$$h(x) = \theta_0 + \theta_1 x$$

Where,

θ_i: Parameter

- Cost function: Find the value of θ_0 and θ_1 such that the half of average of sum of square of difference of predicted and actual value can be minimised:

$$J(\theta_0, \theta_1) = \min_{(\theta_0,\theta_1)} \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^i - y^i)^2 = \min_{(\theta_0,\theta_1)} \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - y^i)^2$$

Where,

m: No. of training examples

h: hypothesis function

$h(x^i)$: prediction for x^i

θ_i: i^{th} parameter

y^i: actual value for x^i

$h(x^i) = \theta_0 + \theta_1 x^i$

1/(2m): Avg. & simplify the math

$\hat{y}^i$: predicted value

- $J(\theta_0, \theta_1)$ is a cost function, known as Squared Error Function or Mean Squared Error (MSE). It is one of the most commonly used cost function for regression problems.
- The mean is halved as a convenience for the computation of the gradient descent as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

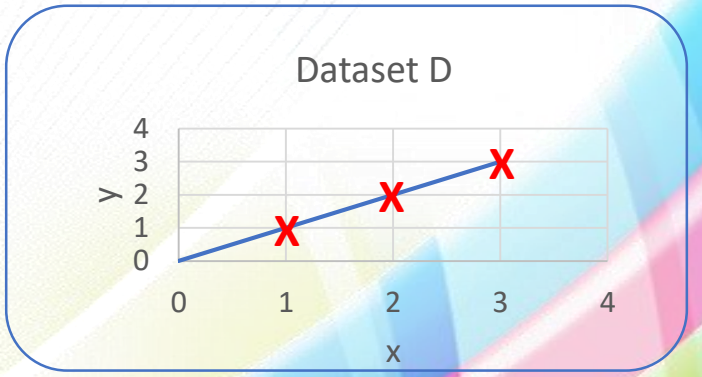
Model and Cost Function

Cost Function Intuition – I

- We have a dataset D with total number of instances, m = 3

Dataset D

Sr. No.	x	y
1	1	1
2	2	2
3	3	3



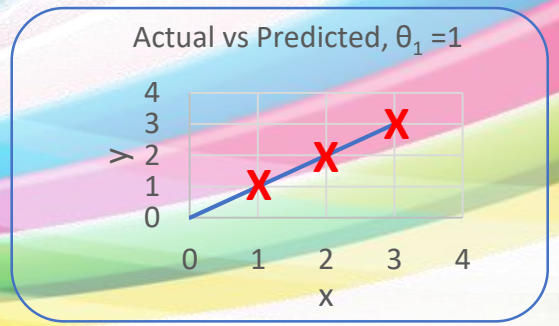
- Let's assume that the hypothesis function is: $h(x) = \theta_1 x$. Thus,

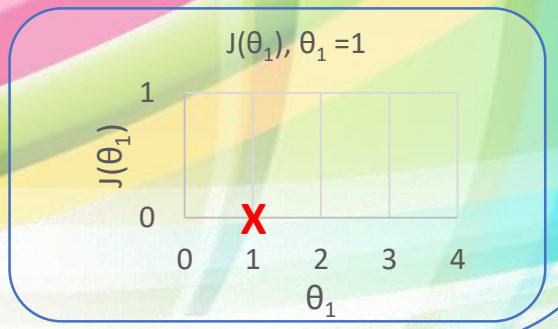
$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(\theta_1 x^i - y^i\right)^2$$

For $\theta_1 = 1$,

$h(1) = 1, h(2) = 2, h(3) = 3$

$J(1) = \frac{1}{6}[(1 - 1)^2 + (2 - 2)^2 + (3 - 3)^2] = 0$





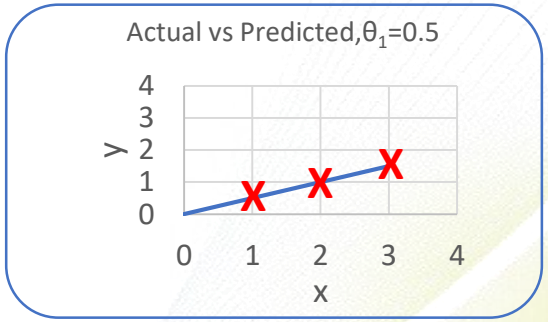
Model and Cost Function

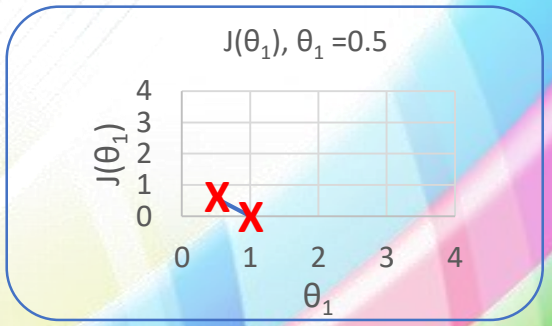
Cost Function Intuition – I

For $\theta_1 = 0.5$,

$h(1) = 0.5, h(2) = 1, h(3) = 1.5$

$J(1) = \frac{1}{6}[(0.5 - 1)^2 + (1 - 2)^2 + (1.5 - 3)^2] = 0.58$

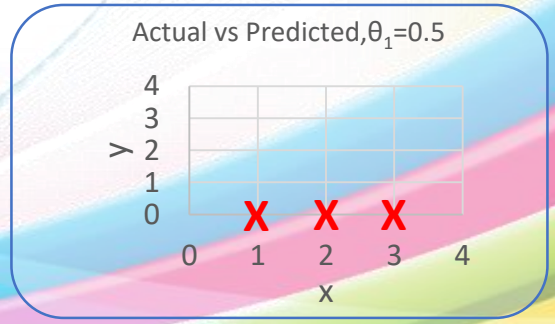


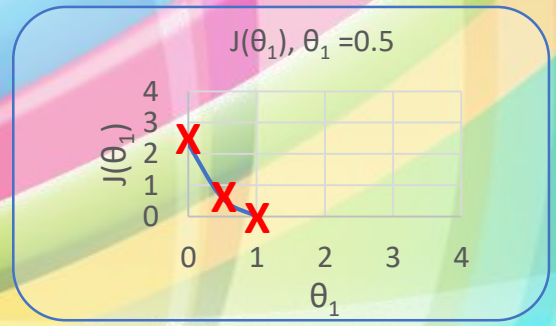


For $\theta_1 = 0$,

$h(1) = 0, h(2) = 0, h(3) = 0$

$J(1) = \frac{1}{6}[(0 - 1)^2 + (0 - 2)^2 + (0 - 3)^2] = 2.33$

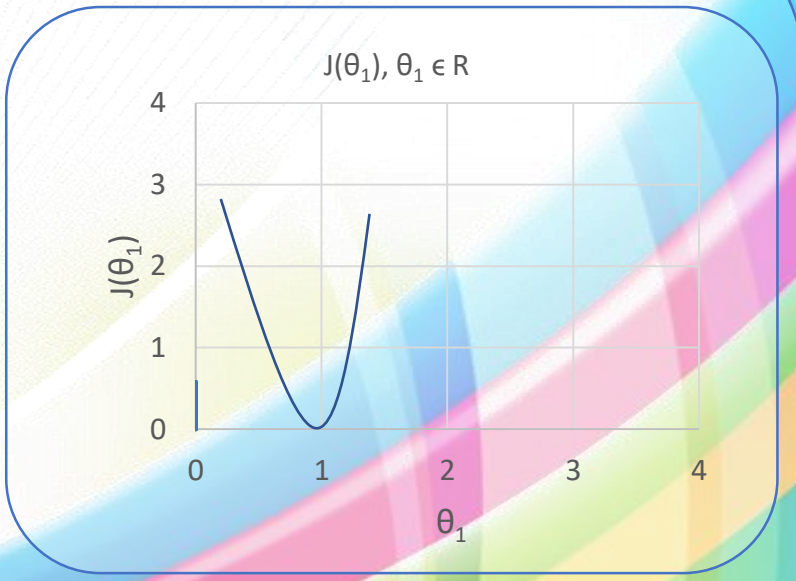




Model and Cost Function

Cost Function Intuition – I

- Hence, when we play more with the value of θ_1, we will get a plot something like depicted next to this.
- This plot shows that for every cost function $J(\theta_1)$, there is a value of θ_1 at which the value of $J(\theta_1)$, i.e., cost, becomes minimum or zero.
- Thus, we should to minimize the cost function. In this case, $\theta_1 = 1$, is our global minimum.



Model and Cost Function

Cost Function – Intuition II

- In case of cost function, what we know so far is following:

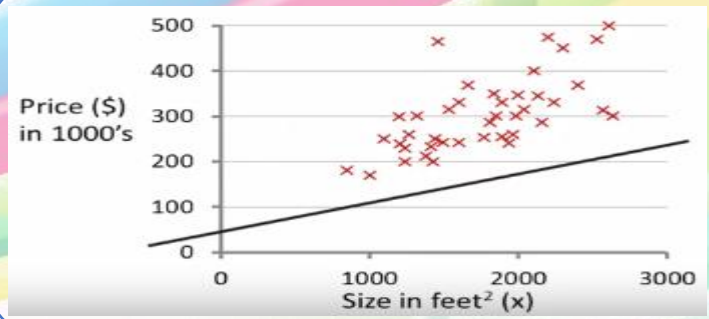
Hypothesis: $h(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)^2$

Goal: $\min_{(\theta_0,\theta_1)} J(\theta_0, \theta_1)$

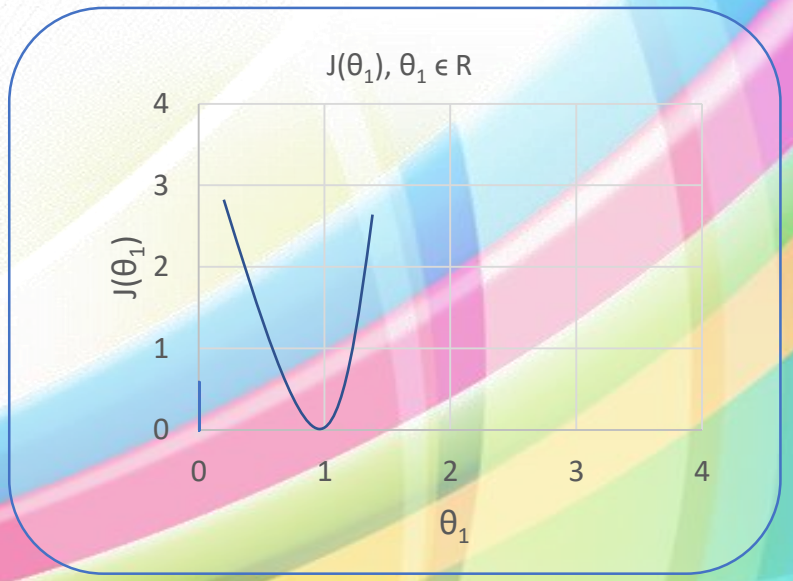
- Unlike "Cost Function – Intuition I" (last slides), this time we will use both the parameters, θ_0 & θ_1.
- Suppose we have a housing price data set and the value of our parameters is $\theta_0 = 50$ & $\theta_1 = 0.06$. Then, the plot of our data set looks something like this:



Model and Cost Function

Cost Function – Intuition II

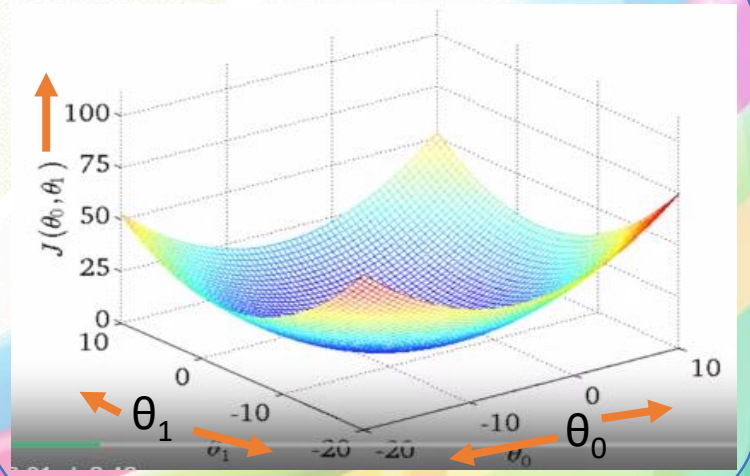
When we had only one parameter, θ_1 in "Cost Function – Intuition I", the plot of our cost function was like this (bowl shaped):



Model and Cost Function

Cost Function – Intuition II

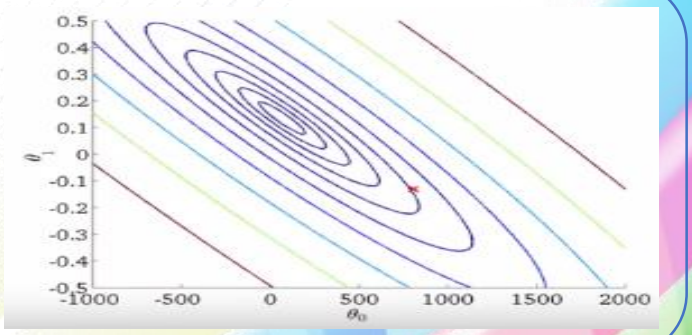
In case of two parameters, θ_0 & θ_1, the shape of cost function will become something like this (bowl shaped surface) depending on your dataset:

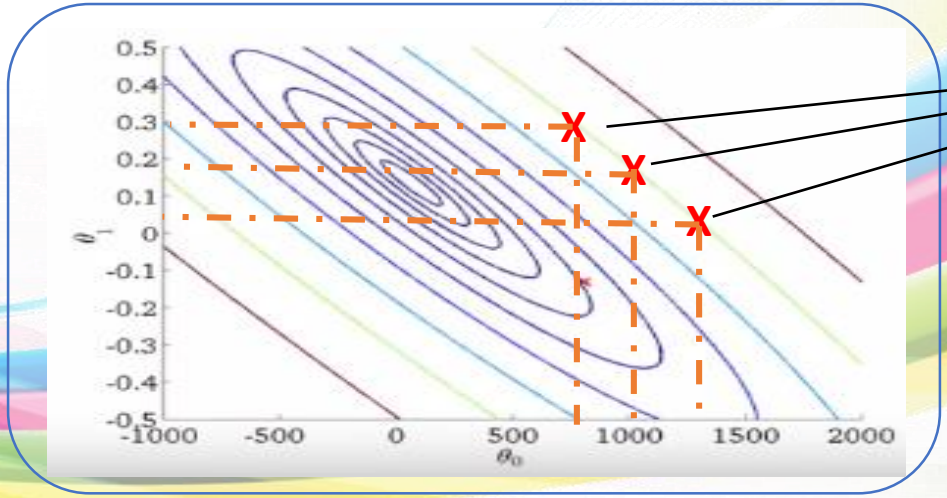


Model and Cost Function

Cost Function – Intuition II

- Instead of making 3D plots of parameters and cost function, like earlier, we will mostly create Contour Plots or Contour Figures.
- In the following plot, θ_0 is on x-axis & θ_1 is on y-axis.
- Each ellipse inside the plot is showing that $J(\theta_0, \theta_1)$ is getting same value as the height of ellipse is same from the two axes.
- Centre of all concentric ellipse is the minimum of $J(\theta_0, \theta_1)$.
- For example:



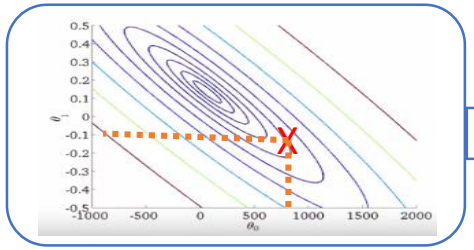


All these three points have the same value for $J(\theta_0, \theta_1)$ as they are on the same ellipse, but the values of θ_0 & θ_1 are different for all three.

Model and Cost Function

Cost Function – Intuition II

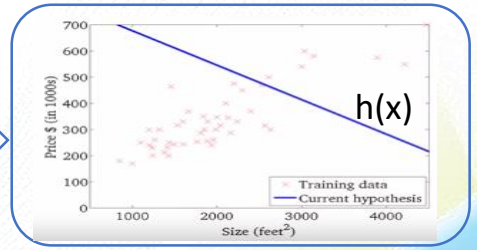
- Let us consider different values for θ_0 & θ_1 on the contour and see the plot of corresponding hypothesis (size of flat on x-axis and price is on y-axis, for every value of θ_0 & θ_1 :



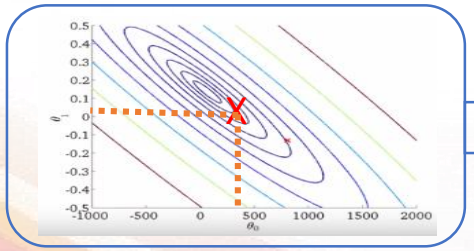
$\theta_0 = 800, \theta_1 = -0.1$

$y^i = h(x^i) = \theta_0 + \theta_1 x^i$

$y^i = h(x^i) = 800 - 0.1x^i$



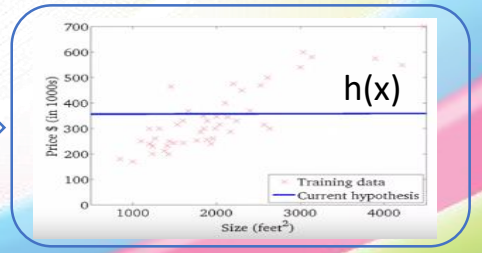
Here, the plot of h(x) is extremely inaccurate due to unoptimized values of θ_0 & θ_1.



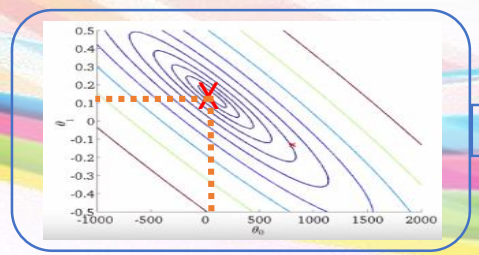
$\theta_0 = 360, \theta_1 = 0$

$y^i = h(x^i) = \theta_0 + \theta_1 x^i$

$y^i = h(x^i) = 360 + 0x^i$



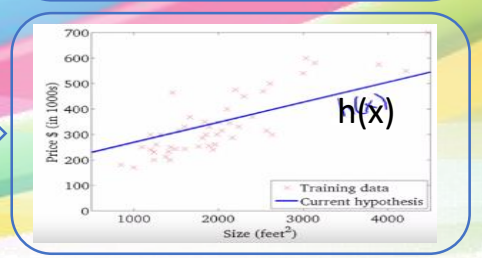
Since θ_1 is zero here, hence for any value of x, h(x) will always be θ_0, that is 360.



$\theta_0 = 250, \theta_1 = 0.15$

$y^i = h(x^i) = \theta_0 + \theta_1 x^i$

$y^i = h(x^i) = 250 + 0.15x^i$



Here, the line of h(x) is quite accurate. Thus, the error is very less for these values of θ_0 & θ_1.

Parameter Learning

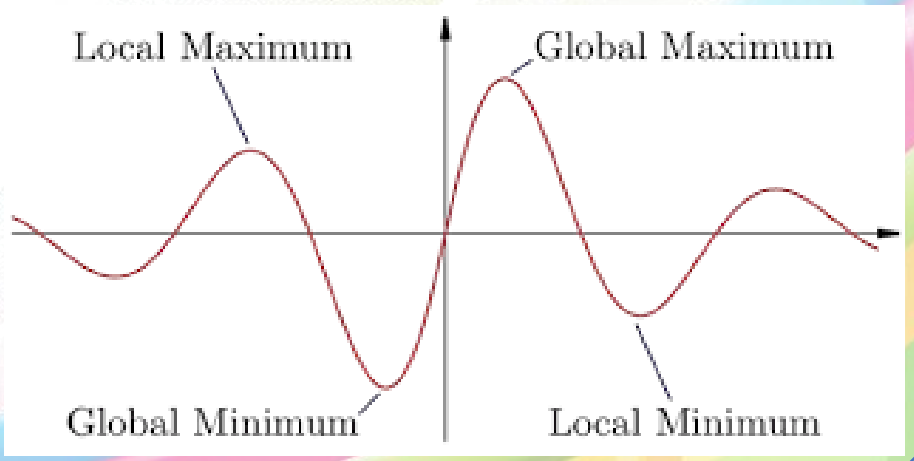
Gradient Descent: Prerequisite

Local Minimum / Maximum

- To get a local minimum or maximum, we need to have an interval.
- The local minimum or maximum is the point in the plot of a function which is at the lowest / highest position within that interval.

Global / Absolute Minimum / Maximum

- Global minimum or maximum is also called as absolute minimum or maximum
- This is the point in the entire plot of a function which is at the highest or lowest position in the entire plot.
- A global min. / max. can be only one.



Parameter Learning

Gradient Descent

- Gradient Descent is an algorithm to minimize the cost function $J(\theta_0, \theta_1)$.
- Gradient Descent is used everywhere in Machine Learning.
- First, we will use Gradient Descent to minimize some random function and later, we will use it to minimize some cost function.
- Following are the background and outline of our objective:

  Some function: $J(\theta_0, \theta_1)$

  Objective: $\min_{(\theta_0,\theta_1)} J(\theta_0, \theta_1)$

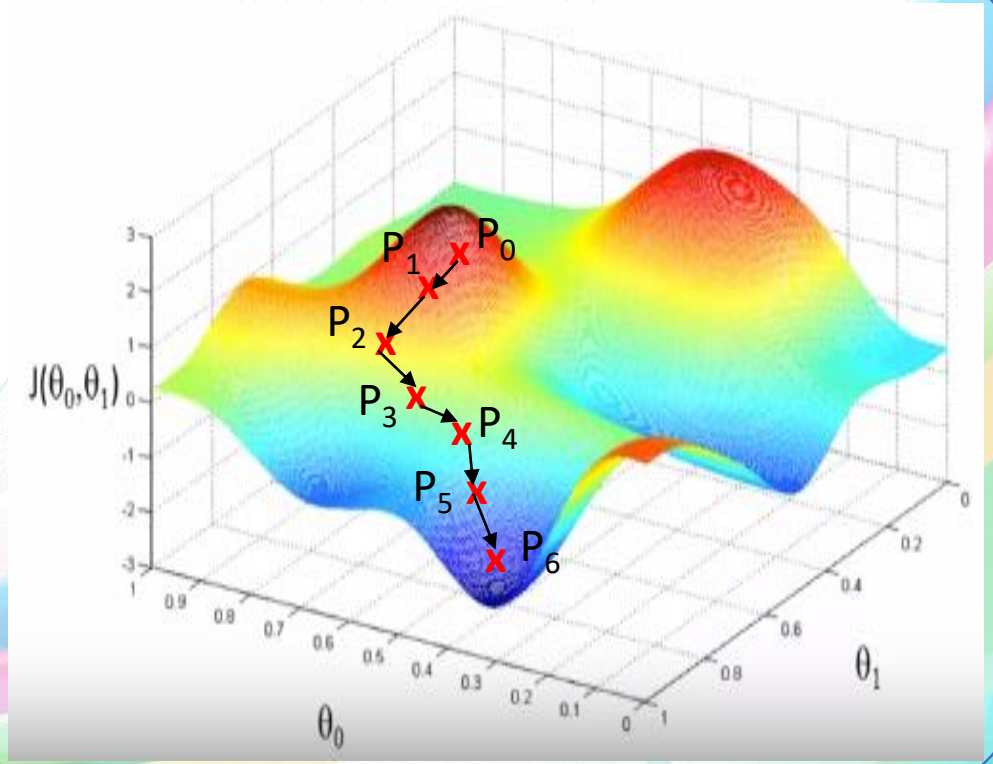
  Outline:
  - Start with some θ_0 & θ_1
  - Keep changing θ_0 & θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum
- Gradient Descent can be applied on general functions like: $J(\theta_0, \theta_1, \theta_2, \ldots\ldots, \theta_n)$
- To understand this, we will consider only two parameters: θ_0 & θ_1.
- A common choice to initialize the value of parameters is 0, i.e., $\theta_0 = 0$ & $\theta_1 = 0$.
- There are chances that by applying Gradient Descent we may end up at a local minimum.

Parameter Learning

Gradient Descent

What Gradient Descent does?

- Suppose you have initialised θ_0 & θ_1 with some random value (that can be 0, as well), due to which you are at position P_0 in the shown graph.
- Now, Gradient Descent will help us to find out the direction in which the value of our cost function $J(\theta_0, \theta_1)$ will reduce most by moving a tiny step (or baby step ☺) in that direction.
- Consequently, we will reach at P_1. Again, Gradient Descent will help us to find a direction when the value of $J(\theta_0, \theta_1)$ will reduce most. Again, we will advance by a tiny step in that direction.
- On repeating this many more times, we will reach at point P_6 which is our local minimum, i.e., the minimum cost.

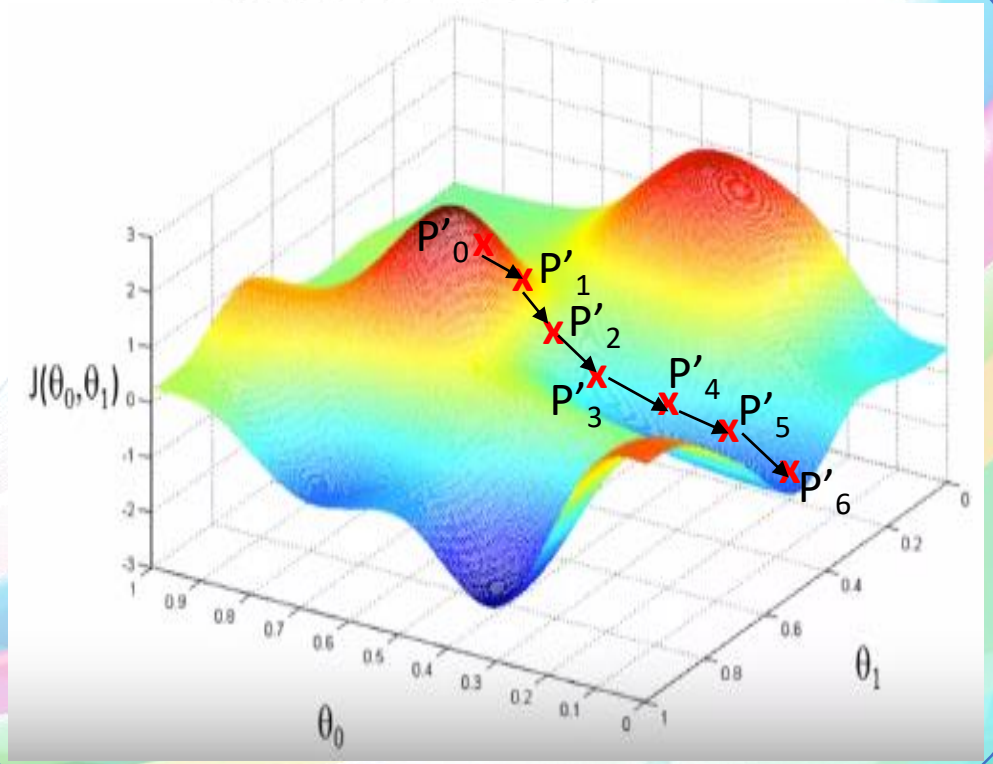


Parameter Learning

Gradient Descent

What Gradient Descent does?

- Suppose, this time we have initialised θ_0 & θ_1 with some other random value, due to which you are at position P'_0 in the shown graph.
- Now, Gradient Descent will help us to find out the direction in which the value of our cost function $J(\theta_0, \theta_1)$ will reduce most by moving a tiny step (or baby step ☺) in that direction.
- Consequently, we will reach at P'_1. Again, Gradient Descent will help us to find a direction when the value of $J(\theta_0, \theta_1)$ will reduce most. Again, we will advance by a tiny step in that direction.
- On repeating this several times, we will reach at point P'_6 which is our local minimum, i.e., the minimum cost, which is different than last time. This is a property of Gradient Descent.



Parameter Learning

Gradient Descent

- Following is the definition of Gradient Descent:

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(For j = 0 and j = 1)}$$

}

Following is the correct simultaneous update of above function:

$$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \qquad \text{(for j = 0)}$$

$$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \qquad \text{(for j = 1)}$$

$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

Above update steps should be follow in the same order, otherwise change in order can raise a blunder error.

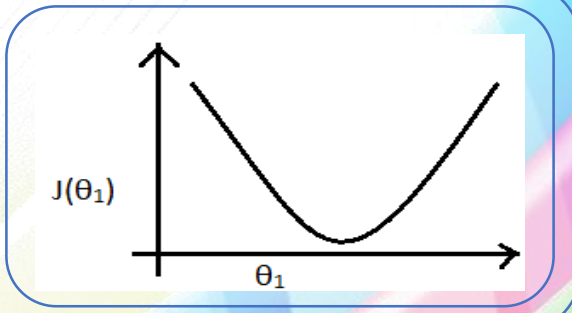
α: This is the learning rate. It controls the length of our step by which we will descend in a particular direction. More on this will come later.

$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$: This is a derivative term that we will discuss later in more detail.

Parameter Learning

Gradient Descent

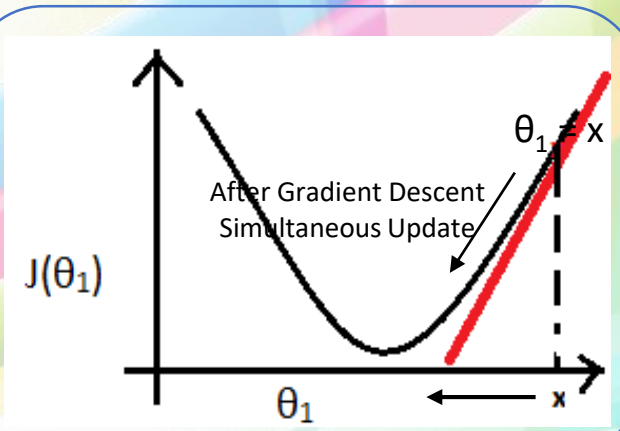
- Let us consider an example of one variable: θ_1
- Suppose, following is the graph of $J(\theta_1)$:



- According to Gradient Descent Simultaneous Update:

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

- As we know that $\frac{\partial}{\partial \theta_1} J(\theta_1)$ is the derivative term..
- The job of derivative term is to provide the slope of tangent at any θ_1 (say $\theta_1 = x$).
- Since, in the shown graph, the slope of line at $\theta_1 = x$, is positive, hence the value of overall Gradient Descent Simultaneous Update will decrease and take θ_1 towards the minimum.



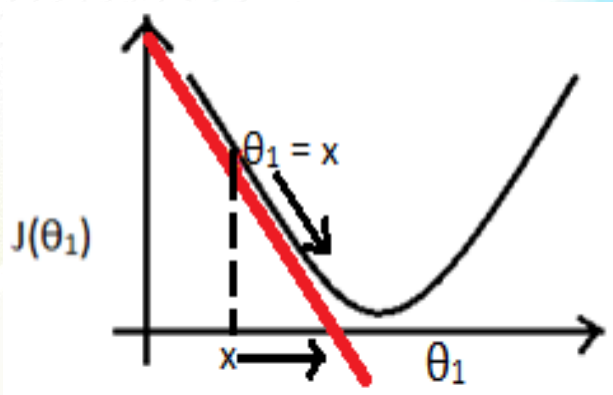
Parameter Learning

Gradient Descent

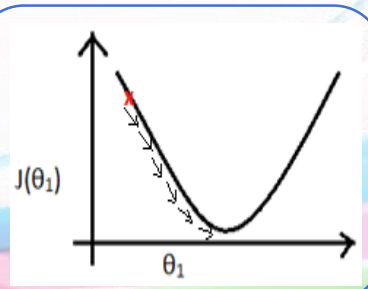
- According to Gradient Descent Simultaneous Update:

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

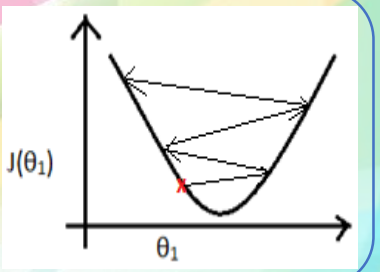
- As we know that $\frac{\partial}{\partial \theta_1} J(\theta_1)$ is the derivative term..
- The job of derivative term is to provide the slope of tangent at any θ_1 (say $\theta_1 = x$). This, time $\theta_1 = x$ is at left side of plot, as shown in the graph.
- Since, in the shown graph, the slope of line at $\theta_1 = x$, is negative, hence the value of overall Gradient Descent Simultaneous Update will increase and take θ_1 towards the minimum.



If learning rate (α) is too small, then gradient descent can be slow.



If learning rate (α) is too large, then gradient descent can overshoot the minimum. It may fail to converge or even diverge.

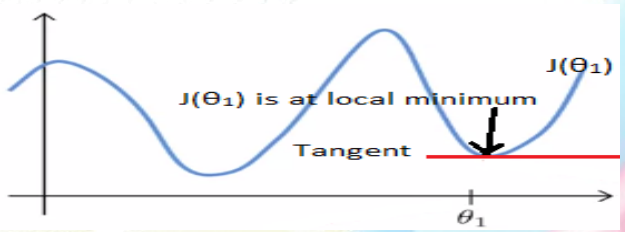


Parameter Learning

Gradient Descent

When θ_1 is already at its local minimum, then the slope of tangent would be 0, that will make the derivative term zero, hence the value of θ_1 remain unchanged.

$$\theta_1 := \theta_1 - \alpha\, 0$$
$$\theta_1 := \theta_1$$

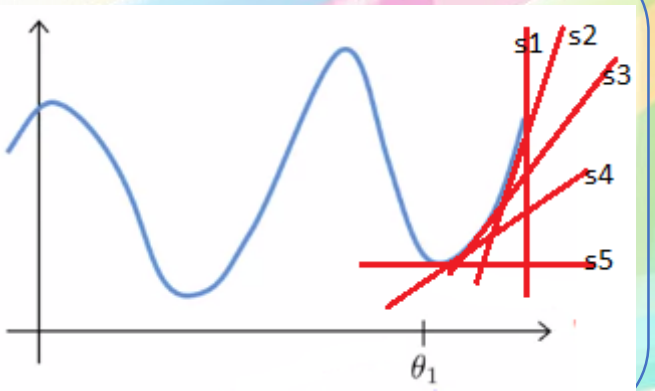


Gradient Descent can converge to a local minimum, even with the learning rate (α) fixed.

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

As θ_1 reaches close to its local minimum, the slope of tangent will reduce and make the overall product of learning rate (α) and the derivative term smaller and smaller, resulting in smaller change in θ_1. Thus, no need to decrease α over time.

Thus, the slope of tangent is in this order: s1 > s2 > s3 > s4 > s5.



Parameter Learning

Gradient Descent for Linear Regression

- So far, we have covered following two things:

Gradient Descent Algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(For j = 0 and j = 1)

}

Linear Regression Model

Hypothesis: $h(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right)^2$

- Now, we will apply Gradient Descent algorithm to minimize the cost function of Linear Regression model, that is:

$$\min_{(\theta_0, \theta_1)} J(\theta_0, \theta_1)$$

- The partial derivative term of Gradient Descent plays a key role in achieving the above objective, that is, minimising cost function.
- In the next slide, we will see the maths of this.

Parameter Learning

Gradient Descent for Linear Regression

- Following is the maths required to minimise the cost function:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^i - y^i\right)^2$$

Now, the partial derivative for j = 0 and j = 1 is:

For j = 0:
$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right)$$

For j = 1:
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right) . x^i$$

By learning the partial derivative, one can easily solve and find the above equations.

Thus, the simultaneous update of Gradient Descent will look like:

$$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right)$$

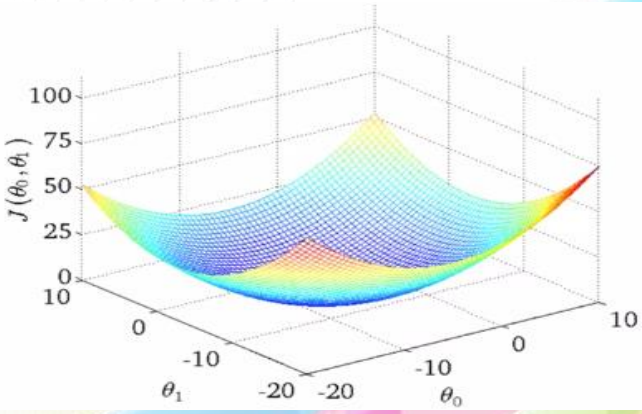
$$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right) . x^i$$

$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

Parameter Learning

Gradient Descent for Linear Regression



- The plot of cost function of Linear Regression is actually a bowl shaped function (see the following image), formally called as “Convex Function”.

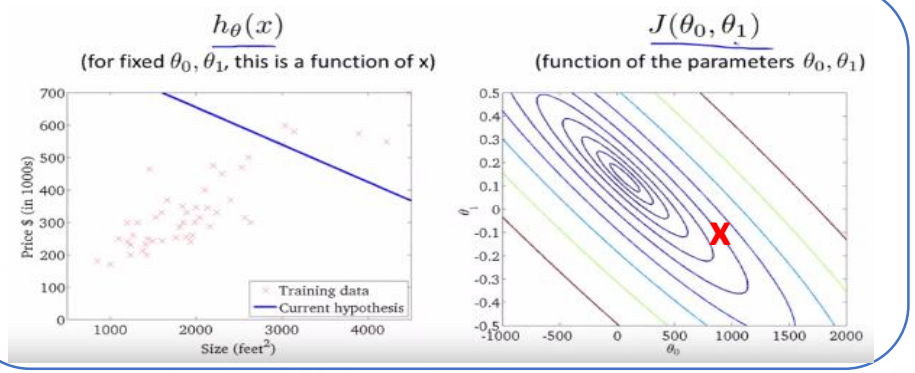
- One property of Convex functions is that they always have one local minima which coincides with the global minima.

Parameter Learning

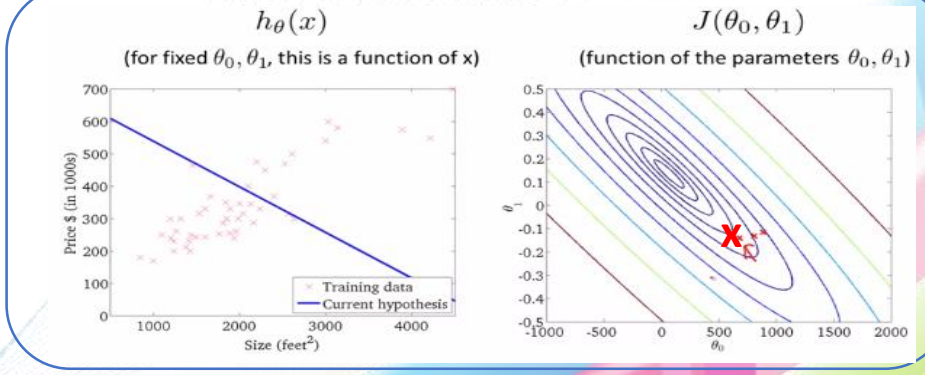
Gradient Descent for Linear Regression

- Following are some of the iterations with different values θ_0 and θ_1. Generally, we start with $\theta_0 = 0$ and $\theta_1 = 0$, but here we have started with some random values.

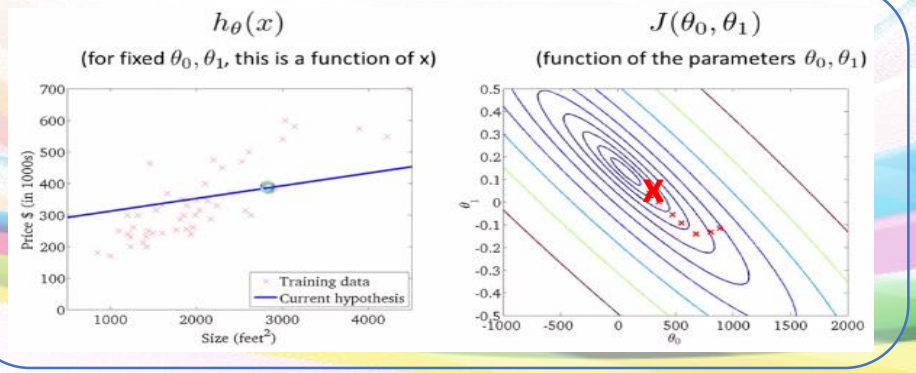
1



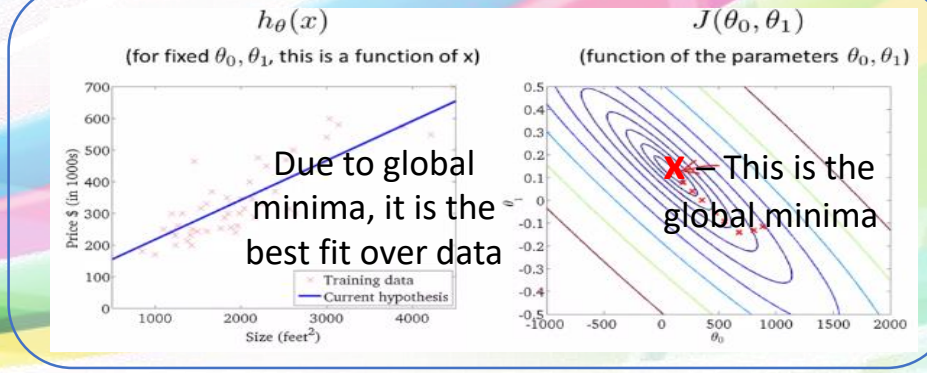
2



3



4



Parameter Learning

Gradient Descent for Linear Regression

- The Gradient Descent algorithm that we have seen so far is specifically called as Batch Gradient Descent.
- Batch: Each step of Gradient Descent uses all the training instances.
- Quiz:

  Which of the following are true statements? Select all that apply.

  1. To make Gradient Descent converge, we must slowly decrease α over time.
  2. Gradient Descent is guaranteed to find the global minimum for any function (θ_0, θ_1).
  3. Gradient Descent can converge even if α is kept fixed, but α cannot be too large, or else it may get fail to converge.
  4. For the specific choice of cost function $J(\theta_0, \theta_1)$ used in linear regression, there are no local optima (other than global optimum).

  Ans: 3 & 4
- Gradient Descent is an iterative algorithm as it computes the value of its parameters (θ_0, θ_1) again and again, and converged. However, in the Advanced Linear Algebra, there are methods to do the same thing, i.e., finding minimum of parameters (θ_0, θ_1), in one step. We will look at those methods later.
- That Advanced Linear Algebra method is Normal Linear Equation method, but Gradient Descent is better than that as Gradient Descent scales better on huge data set than Normal Linear Equation method.

Linear Algebra Review

- Following are the topics covered under this heading:

1. Matrices and Vectors
2. Addition and Scalar Multiplication
3. Matrix Vector Multiplication
4. Matrix Matrix Multiplication
5. Matrix Multiplication Properties
6. Inverse and Transpose

Multivariate Linear Regression

Multiple Features

- Earlier, we considered Linear Regression with single feature. In that case, data was as shown in figure:
- In this case, h(x) is the hypothesis function.

Size (feet²)	Price ($1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

$$h_\theta(x) = \theta_0 + \theta_1 x$$

- Here, we will consider Linear Regression with multiple features.
- In this case, {size, number of bedrooms, number of floors, age of home} are the features and {price} is the target variable.
- We will use {x_1, x_2, x_3, x_4} to represent features and {y} to represent the target variable, "price".

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Multivariate Linear Regression

Multiple Features

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

- Notations:
  - n: Number of features, here it is 4.
  - x^i: All input features of i^{th} training example
  - x^i_j: Value of feature j in i^{th} training example
  - m: Total number of training examples
- Thus, $x^{=} = [1416, 3, 2, 40]$ and $x^2_3 = 2$
- Earlier, in case of single features, our hypothesis function was:

$$h(x) = \theta_0 + \theta_1 x$$

- However, now in case of multiple features, our hypothesis function will be (for n = 4):

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

- This can be generalised as:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta_0 + \sum_{k=1}^{n} \theta_k x_k$$

- Another form of writing the above equation:

Suppose, $x = [x_0, x_1, x_2, ..., x_n]$, here $x_0 = 1$ (always) and $\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ \theta_n \end{pmatrix}$, $\theta^T = [\theta_0, \theta_1, ..., \theta_n]$. Thus, $h(x) = \theta^T x$. This is called as **Multivariate Linear Regression**.

Multivariate Linear Regression

Gradient Descent for Multiple Features

- Now, we have following things:
  - Hypothesis function: $h(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$, where $x_0 = 1$.
  - Parameters: $\{\theta_0, \theta_1, \theta_2, \ldots, \theta_n\}$, let us call it simply θ.
  - Cost function: $J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)^2$
- Now, following is the algorithm of Gradient Descent for multiple features:

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_j$$

Simultaneously update θ_j for j = 0, 1, 2, …, n.

Here, α is the learning rate.

Above term, i.e., $\frac{1}{m}\sum_{i=1}^{m}(h(x^i) - yi)x^i{}_j$, is actually the partial derivative of cost function $J(\theta)$, i.e., $\frac{\partial}{\partial\theta_j}J(\theta)$.

Compare it with the Gradient Descent equation for single variable and comprehend it properly.

Thus, if we have two features, then:

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_0$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_1$$

$$\theta_2 := \theta_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_2$$

Where, $x_0 = 1$.

Multivariate Linear Regression

Gradient Descent in Practice I – Feature Scaling

- Here, we will discuss a few practical tricks to make Gradient Descent work well.
- One of such tricks is Feature Scaling.
- Feature Scaling: make sure that features are on a similar scale.

  For example:

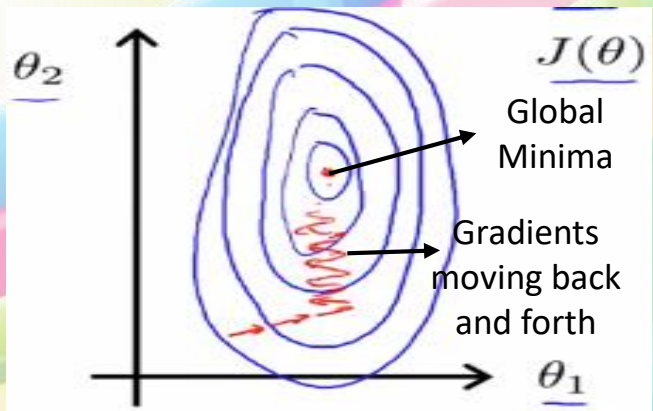
  x_1: size of apartment (0 – 2000 sq. feet)

  x_2: number of bedrooms (1 – 5)

  The cost function, J(θ), for above features will have three parameters, $J(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

  Let us ignore θ_0 for a while and see the plot of θ_1 and θ_2:

  - In this plot, θ_1 is on x-axis and θ_2 is on y-axis.
  - The plot is of cost function J(θ) (i.e., $\theta_1 x_1 + \theta_2 x_2$, θ_0 ignored).
  - Because of wide range of x_1 (i.e., 0 – 2000) and very narrow range of x_2 (i.e., 1 – 5), the contours of J(θ) will be very skewed (i.e., tall and skinny) and elliptical in shape.
  - Due to this, the Gradient Descent algorithm will take a very long time to reach at global minima at the centre as the gradient will be changing back and forth (see zig-zag red arrows).



Multivariate Linear Regression

Gradient Descent in Practice I – Feature Scaling

- The previous issue of taking a very long time for convergence can be resolved by scaling the features.
- This can be done as:

  Our actual features:

  x_1: size (0 – 2000 square feet)

  x_2: number of bedrooms (1 – 5)

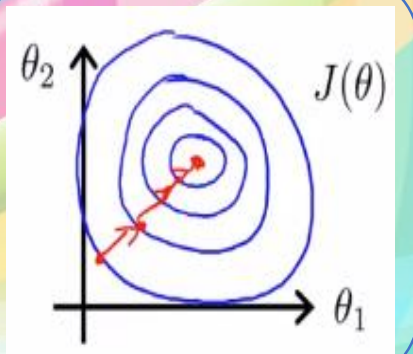
  Now, scaled x_1 and x_2:

$$x_1 = \frac{size\ (feet^2)}{2000} \quad \text{and} \quad x_2 = \frac{number\ of\ bedrooms}{5}$$

  The above mathematical operation will result into: $(0 \le x_1 \le 1)$ & $(0 \le x_2 \le 1)$.

  In this case, the contours of cost function J(θ), i.e., $\theta_1 x_1 + \theta_2 x_2$ (ignore θ_0), will look like this:

  - The contours of J(θ) will become less skewed (i.e., tall and skinny) or may be circular.
  - Due to this, the Gradient Descent will perform substantially better.
  - And Gradient Descent algorithm would be able to reach at the centre, i.e., Global minima, quite earlier. That is, the convergence will be much faster.



Multivariate Linear Regression

Gradient Descent in Practice I – Feature Scaling

- Generally, when we do feature scaling, we often want our features to be in the range of -1 and +1:

$$-1 \le x_i \le +1$$

- The number -1 and +1 are not important. This can be possible that the range of a feature after scaling is 0 &1 or 0 & +3 or -2 & +0.5, etc. Since these ranges are quite close to -1 & +1 range, hence they are fine.
- However, if the ranges are substantially higher or lower than -1 & +1, such as:

$$-100 \le x_i \le +100 \quad \text{or} \quad -0.0001 \le x_i \le +0.0001$$

  Then, this is called as poorly scaled feature.
- Different people have different thumb rule for the accepted range after feature scaling. One such rule that has been accept by Andrew Ng is around -3 to +3 or -0.333 to +0.333.
- Sometime, people also take another approach of scaling, called as Mean Normalization.
- Mean Normalization:
  - Replace x_i with $(x_i - \mu_i)$, where μ_i is the average value of x_i, to make features have approximately zero mean (a data with mean 0 as standard normal distribution has mean 0 and variance 1).
  - Do not do it for x_0, as $x_0 = 1$.
  - For instance, when use both Mean Normalization and Feature Scaling, i.e., $(x_i - \mu_i) / (\max(x_i) - \min(x_i))$:

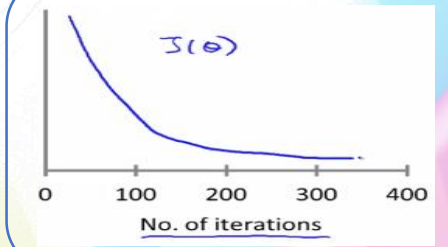
$$x_1 = \frac{size - 1000}{2000} \quad \text{and} \quad x_2 = \frac{\#bedrooms - 2}{5}$$

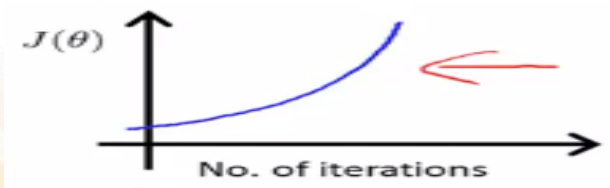
This will result in: $-0.5 \le x_1 \le +0.5$ and $-0.5 \le x_2 \le +0.5$

Multivariate Linear Regression

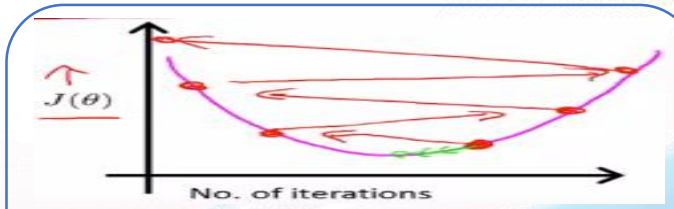
Gradient Descent in Practice II – Learning Rate

- In the corresponding graph, number of training iterations is on x-axis and the value of cost function J(θ) is on y-axis.
- To make sure that the gradient descent is working fine, we should plot the values of J(θ).
- If the plot is declining with each iteration of training, then it means that the gradient descent is working fine and cost is reducing.
- Thus, J(θ) should decrease after each iteration.

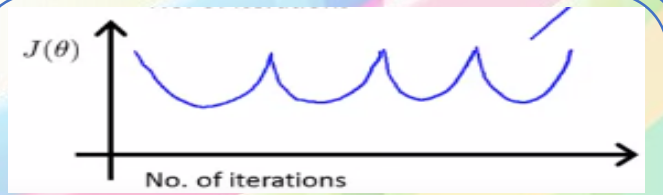




- If this plot of J(θ), cost is increasing.
- This can be resolved by further reducing learning rate



- Here, gradients are overshooting due to higher learning rate.
- Thus, reduce learning rate.



- If the plot of J(θ) is weird like above, then also reduce the learning rate.

- For sufficiently small learning rate (α), J(θ) should decrease on every iteration
- But, if learning rate (α) is too small, gradient descent can be extremely slow to converge.

Multivariate Linear Regression

Features and Polynomial Regression

- Suppose, we have two features for house pricing prediction problem:
  i. Frontage
  ii. Depth
- The hypothesis function $h(x_1, x_2)$ for this problem can be like this:

$$h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

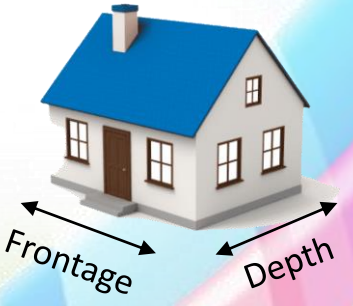
  Where, x_1: frontage and x_2: depth
- Above we have two features, but we can create a new feature by combining them.
  For instance:

$$\text{area, } x = x_1 * x_2 = \text{frontage x depth}$$

  Thus, our new hypothesis function h(x) would look like this:

$$h(x) = \theta_0 + \theta_1 x$$

- Sometimes, by defining new features, we may get a better model.



Multivariate Linear Regression

Features and Polynomial Regression

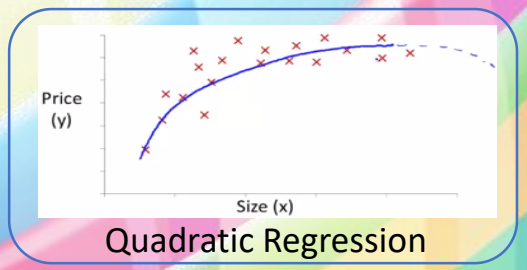
- In the given graph, we have size (i.e., area = frontage x depth) of house on x-axis.
- Price is on the y-axis.
- Apparently, it is clear from the plot that a linear regression cannot fit this.
- Thus, we have to use a polynomial regression.
- In order to perform that, suppose first we have chosen the quadratic model.
- Following are the equations of linear regression model and quadratic model:

  Linear Regression: $h(x) = \theta_0 + \theta_1 x$

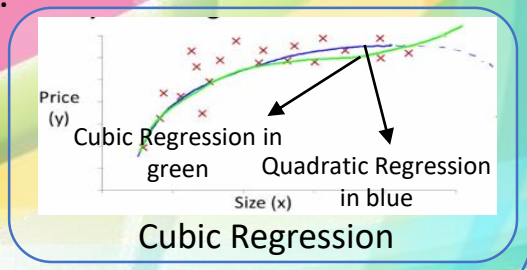
  Quadratic Regression: $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

  Where, x: area
- The Quadratic Regression model will fit in the beginning, but later it will not.
- It is shown in the graph "Quadratic Regression" that the model will decline in later stages.
- However, we know that the prices will increase further in later stages.
- To rectify this, we can use a cubic function or cubic regression, as shown in the figure.
- Cubic regression: $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- To implement quadratic or cubic regression, create three variables in data set:

  x_1: x (i.e., area) x_2: x^2 (i.e., $area^2$) x_3: x^3 (i.e., $area^3$)
- Since ranges of x_1, x_2 and x_3 will get changed, hence do not forget to do feature scaling with their respective ranges.



Quadratic Regression



Cubic Regression

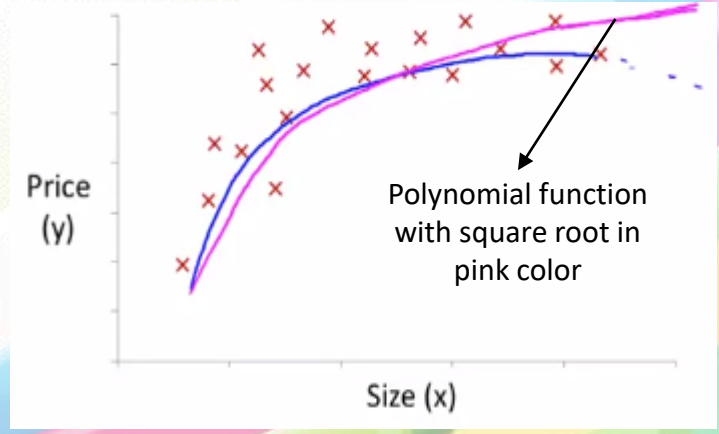
Multivariate Linear Regression

Features and Polynomial Regression

- Sometimes, another type of polynomial function can be more fruitful to fit the data.
- One such example is a polynomial function with a square root:

$$h(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x}$$

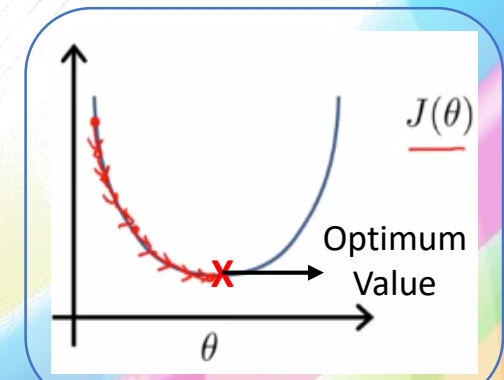
- From the graph, we can see that the plot is increasing slightly in the later stages.



Computing Parameters Analytically

Normal Equation

- Normal Equation gives us much better way to solve for the optimum value of the parameters of θ for some linear regression.
- So far we have seen that the Linear Regression make use of Gradient Descent that works in steps to figure out the optimum values of θs, as shown in the figure.
- However, Normal Equation works analytically to figure out the optimum values of θs in one go, instead of several steps.
- Now, suppose there is a cost function, $J(\theta) = a\,\theta^2 + b\,\theta + c$
- To minimize this cost function, J(θ), we have to take its partial derivative and equate it with zero in order to find the values of θs. Something like as shown below:



$$h(x_1, x_2, \ldots, x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - y^i)^2$$

$$\frac{\delta}{\delta \theta_j} J(\theta) = \ldots\ldots = 0 \text{ (for every j)}$$

Solve for $\theta_0, \theta_1, \theta_2, \ldots, \theta_n$.

Computing Parameters Analytically

Normal Equation

- Unlike the partial derivative method to find the optimised values of θs in several steps to minimise the cost function J(θ), here we will make use of normal equations to achieve the same thing, but in one step.
- Suppose, we have the following data set with total rows, m = 4; and total features, n = 5 (including $x_0 = 1$):

x_0	Size (feet^2) x_1	No. of bedrooms x_2	No. of floors x_3	Age of home (yrs.) x_4	Price ($1,000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

- Create a matrix of features and target values:

$$X = \begin{pmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{pmatrix} \qquad y = \begin{pmatrix} 460 \\ 232 \\ 315 \\ 178 \end{pmatrix}$$

To get values of θs that minimise your cost function J(θ), just solve this: $\theta = (X^T X)^{-1} X^T y$

- In the method of Normal Equation, Feature Scaling is not required. Thus, it doesn't matter if one feature is in range of [0, 1] and another in range of [0, 1000] or [0, 10^{-5}].

Computing Parameters Analytically

Gradient Descent vs Normal Equation

#	Gradient Descent	Normal Equation
1	[Cons] Need to choose learning rate (α)	[Pros] No need to choose learning rate (α)
2	[Cons] Needs many iterations to minimise cost function J(θ)	[Pros] No need to iterate
3	[Pros] Works well even when number of features, n, is very large	[Cons] Since it has to compute $(X^T X)^{-1}$, hence it is extremely slow when number of features, n, is large.

- For smaller value of number of features, i.e., n, Normal Equation will run much faster than Gradient Descent.
- Now, how to figure out that for which value of n (i.e., number of features), Normal Equation will be extremely slow and it would be better to choose Gradient Descent. Following are some guidelines from Andrew Ng:

  In today's modern computers, we can use Normal Equation for value of n up to 10,000. Since the runtime complexity of n in case of Normal Equation becomes $O(n^3)$, hence for number of features greater than 10,000, it would be better to use Gradient Descent.

Computing Parameters Analytically

Normal Equation Non-invertibility

- We know that the Normal Equation has this formula to calculate the optimised values of θs:

$$\theta = (X^T X)^{-1} X^T y$$

Now, what if $(X^T X)$ is non-invertible, i.e., it is a singular matrix?

In Octave, there are two functions to calculate the inverse of any matrix:

i. pinv: sudo-inverse function. This function will provide the values of θ even if $(X^T X)$ is non-invertible.
ii. inv: inverse function. This function will not work if $(X^T X)$ is non-invertible.

Following can be the cause of non-invertibility of $(X^T X)$:

i. Redundant features (linearly dependent): Suppose you have two features which are linearly related to each other, such as, x_1: size in feet^2 and x_2: size in meter^2. Here, $x_1 = (3.28)^2 x_2$ is the relation between x_1 and x_2.
   In such situations, delete one such linearly dependent feature to remove the redundancy.
ii. Too many features: In this case, the number of features (i.e., n) would be much larger than the number of instances (i.e., m). That is, $n \geq m$.
   Two solutions for this case:
   a. Delete some features that are clearly less / not important.
   b. Use regularization.

Computing Parameters Analytically

Octave / Matlab tutorial

Skipped

Topics under this heading

1. Basic Operations
2. Moving Data Around
3. Computing on Data
4. Plotting Data
5. Control Statements – for, while, if
6. Vectorization

Exercise

- Following things have been implemented in Octave:
  1. Linear Regression – Calculation of cost, i.e., J(θ)
  2. Linear Regression – Update value of θs

Classification and Representation

Classification

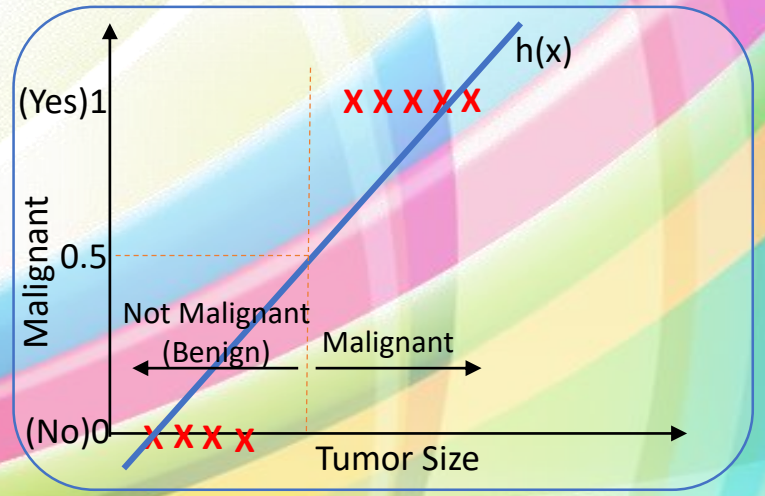
- Following are some of the examples of classification problem:
  - Email: Spam / No Spam
  - Online Transactions: Fraudulent {Yes, No}
  - Tumor: Malignant / Benign
- In all of the above examples, the target variable, y, has taken either value 0 (“negative class”) or 1 (“positive class”).
- Mostly, the meaning of negative and positive class has been considered arbitrarily, nevertheless “negative” means “absence” and “positive” means “presence” of something. Hence,
  - Email:
    - Negative: No Spam
    - Positive: Spam
  - Online Transaction:
    - Negative: Not Fraudulent
    - Positive: Fraudulent
  - Tumor
    - Negative: Benign
    - Positive: Malignant

Classification and Representation

Classification

- Earlier are the examples of Binary Classification where we have only two classes in our target variable, thus:
  $y \in \{0, 1\}$
- Later, we will also see the examples of Multiple Class Classification where we will have more than two classes, i.e.,:
  $y \in \{0, 1, 2, \ldots.\}$

- In the given figure, we can see that "Malignant" has two values on Y-axis, {1: Yes, 0: No}. Whereas, TumorSize is on X-axis.
- Suppose, we have trained a Linear Regression model for this data set and the plot of hypothesis function, i.e., h(x), is shown in blue colour.
- With some analysis, we can figure out that by using the threshold of 0.5, we can utilise this Linear Regression model to work for this Binary Classification problem where:
  - For any value greater than or equal to 0.5, prediction will be "yes", "Malignant".
  - For any value lower than 0.5, prediction will be "no", "Not Malignant or Benign".

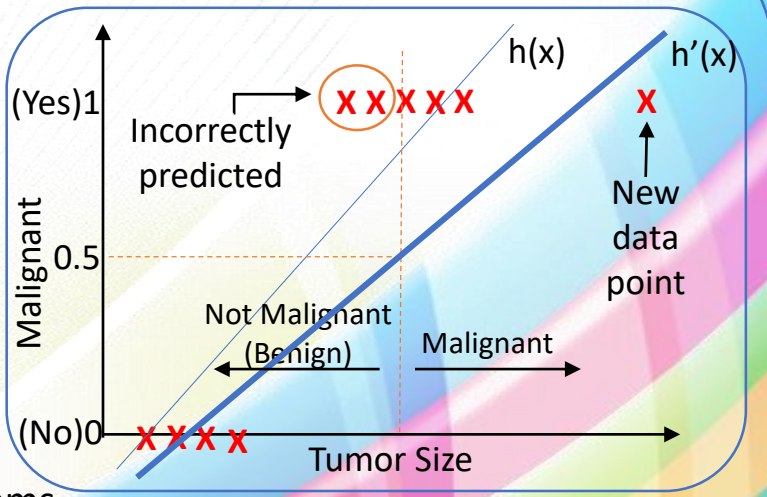


- Thus, here it seems as Linear Regression is working for Classification problems.

Classification and Representation

Classification

- Here, we will see the case in which Linear Regression get failed for Classification problem similar to earlier but with one minor change.
- Now, we have added a “New data point”.
- Due to this, the plot of our hypothesis function get changed and represented by h’(x).
- Now, when we are considering the same threshold, i.e., 0.5, to classify Malignant – Yes or No, we found that it is not working correctly.
- Due to shift in the plot of our hypothesis function, denoted by h’(x), two Malignant points get classified as Not Malignant or Benign.



- Thus, generally Linear Regression cannot be utilised for Classification problems.
- Other than above issue, another issue with the Linear Regression is that, it can output values higher than 1 and lower than 0. However, our classes are represented by only 1 and 0.
- Even if there is a feature x that perfectly predicts y, i.e., $y = 1$ when $x \geq c$ and $y = 0$ when $x < c$ (for some constant c), Linear Regression would not be able to work with 100% accuracy.

Classification and Representation

Hypothesis Representation

- In Logistic Regression, we want our hypothesis function to output values between 0 and 1:

$$0 \leq h(x) \leq 1$$

- The hypothesis function of Logistic Regression looks like following:

$$h(x) = g(\theta^T x)$$

Where,

$$g(z) = \frac{1}{1+ e^{-z}}$$

Here, g(z) is known as Sigmoid Function or Logistic Function.

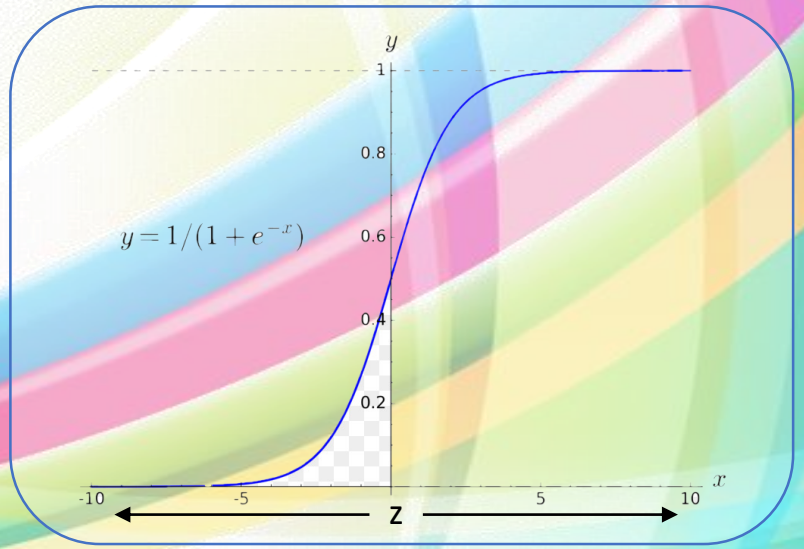
Thus, our hypothesis function is:

$$h(x) = \frac{1}{1+ e^{-\theta^T x}}$$

- Following is the plot of Sigmoid or Logistic function.
- The output of hypothesis function h(x) can be considered as the estimated probability for y = 1 on input x.
- For instance: if h(x) give 0.7, then it means that the model is telling 70% chance of Malignant in the patient. Mathematically,

$$h(x) = P(y = 1 \mid x; \theta)$$

Read as "probability that y = 1, given x parametrized by θ.



Classification and Representation

Decision Boundary

- As we know that:

$$h(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+ e^{-z}}$$

From the plot, it is clear that:

- $g(z) \geq 0.5$ when $z \geq 0$
- $g(z) < 0.5$ when $z < 0$

Thus,

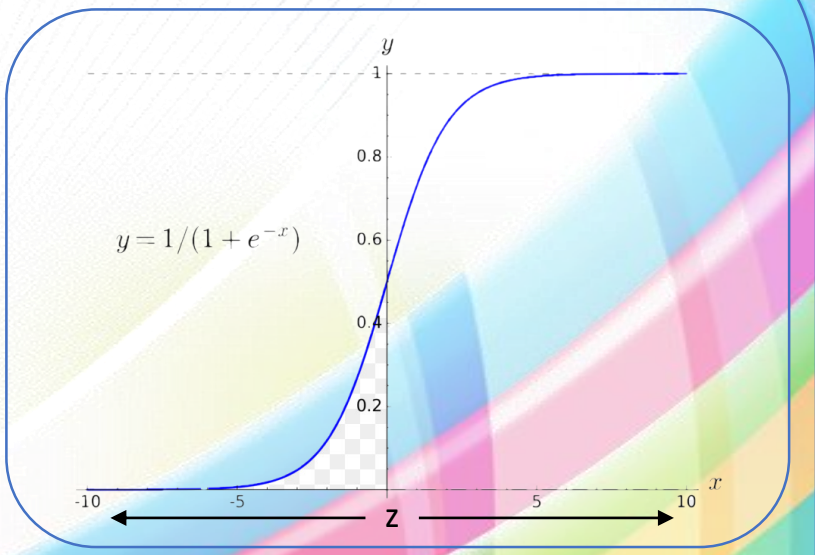
- $h(x) \geq 0.5$ when $(\theta^T x) \geq 0$
- $h(x) < 0.5$ when $(\theta^T x) < 0$

- Suppose:

$$h(x) = g(\theta^T x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Prediction will be:

- 1 when $(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \geq 0$
- 0 when $(\theta_0 + \theta_1 x_1 + \theta_2 x_2) < 0$



Classification and Representation

Decision Boundary

- Assume: $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ 2 \end{bmatrix}$

Then,

$$h(x) = g(\theta^T x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = -3 + x_1 + x_2$$

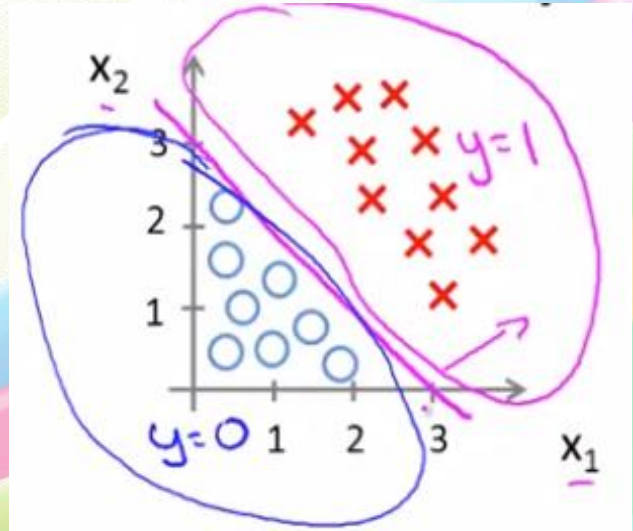
$$-3 + x_1 + x_2 \geq 0 \rightarrow -3 + x_1 + x_2 = 0 \rightarrow x_1 + x_2 = 3$$

This is the equation of a straight line, shown in the purple color passing through 3 on X and Y-axis in the figure.

Since $-3 + x_1 + x_2 \geq 0$ is for y = 1, hence all the points above the purple line x1 + x2 = 3, are malignant.

Whereas, points below line $x_1 + x_2 = 3$ are not malignant or they are benign.

- The purple line in the given figure, passing through 3 on X and Y-axis is called as Decision Boundary.



Classification and Representation

Decision Boundary

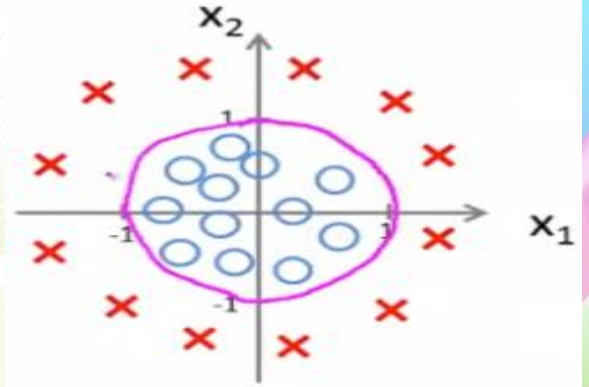
- For the given data set, as shown in the figure, we have to tweak our hypothesis function, h(x).

  Suppose following is our tweaked hypothesis function:

  $$h(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

  Following are the values of our parameters:

  $$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



- Thus, our hypothesis function is $-1 + x_1^2 + x_2^2 \geq 0 \rightarrow x_1^2 + x_2^2 = 1$. This is the equation of a circle.
- Therefore, the Decision Boundary in this case will be a circle as shown in the purple color.
- Inside the circle, the prediction is y = 0 and outside the circle prediction is y = 1.
- Thus, by adding some complicated higher order polynomial features we can make our Logistic Regression to work for complicated data sets.

Logistic Regression Model

Cost Function

- We know that the cost function of Linear Regression is:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^i) - y^i)^2$$

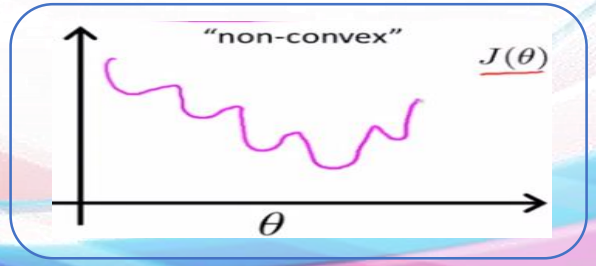
- In case of Linear Regression, the hypothesis function h(x) is a linear function:

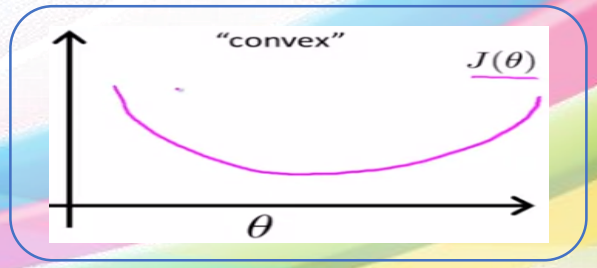
$$h(x) = \theta = + \theta_1 x_1 + \theta_2 x_2 + \ldots. + \theta_n x_n$$

- However, in case of Logistic Regression, the hypothesis function h(x) is a non-linear function:

$$h(x) = \frac{1}{1 + e^{-\theta^T x}}$$

When we insert this non-linear function in the cost function of Linear Regression, it will give us a non-convex function. Consequently, it will not be guaranteed to reach at global minima. Thus, the cost function of Linear Regression cannot use in Logistic Regression.





- Convex Analysis and Optimization is beyond the scope of this course. [DO IT]

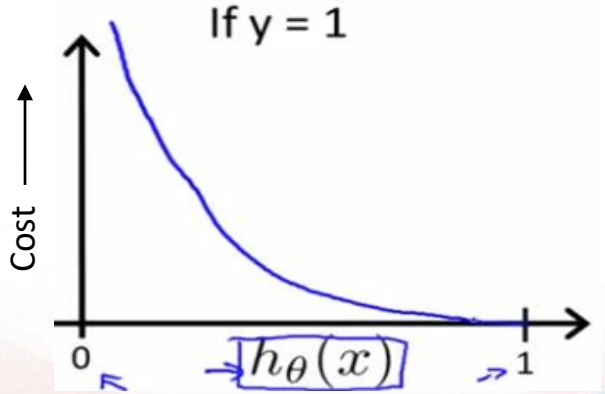
Logistic Regression Model

Cost Function

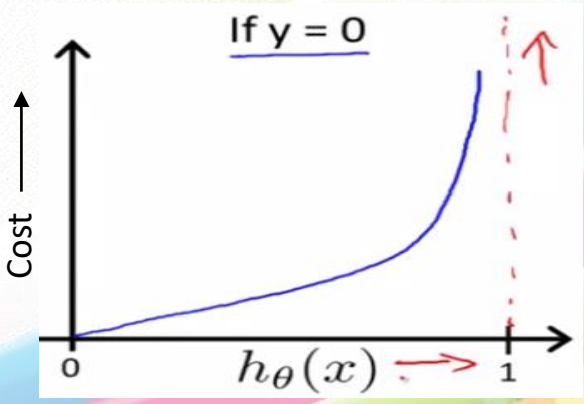
- Following is the cost function that we will use for Logistic Regression:

$$\text{Cost}(h(x), y) = \begin{cases} -\log(h(x)), & \text{if } y = 1 \\ -\log(1 - h(x)), & \text{if } y = 0 \end{cases}$$

- Following are the plots of cost function for y =1 and y = 0:



- Cost = 0, if y = 1 and h(x) = 1
- As h(x) → 0, Cost → ∞
- If h(x) = 0, but y = 1, we will penalize learning algorithm by a very large cost.



- Cost = 0, if y = 0 and h(x) = 0
- As h(x) → 1, Cost → ∞

Logistic Regression Model

Simplified Cost Function and Gradient Descent

- So far, we know following things about Logistic Regression:
  - Cost function, J(θ) = $\frac{1}{m}\sum_{i=1}^{m} Cost(h(x^i), y^i)$
  - Cost(h(x), y) = $\begin{cases} -\log(h(x)), & \text{if } y = 1 \\ -\log(1 - h(x)), & \text{if } y = 0 \end{cases}$
  - Note, y = 0 or y = 1, always.
- Following is a compact and convenient way of writing Logistic Regression's cost function in one line which would be equivalent to above:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h(x^i), y^i)$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log h(x^i) + (1 - y^i)\log(1 - h(x^i))\right]$$

- The above term satisfies the Principle of Maximum Likelihood which is out of scope of this course. [DO IT]
- Now, we want to minimise J(θ). Following is the formula to do this:

Repeat {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j} J(\theta)$$

(Simultaneously update all θ_j)

}

Where, $\frac{\partial}{\partial\theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x_j^i$ α: Learning Rate

Logistic Regression Model

Simplified Cost Function and Gradient Descent

- To figure out that the learning rate (α) is set properly and gradient descent is running correctly:
  - Plot $J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^i \log h(x^i) + (1 - y^i) \log(1 - h(x^i))]$ as a function of the number of iterations and make sure that J(θ) is decreasing on every iteration.
- The vectorized implementation of Gradient Descent is:

$$\theta := \theta - \alpha\frac{1}{m}\sum_{i=1}^{m}[(h(x^i) - y^i).x^i]$$

For all θs.

Logistic Regression Model

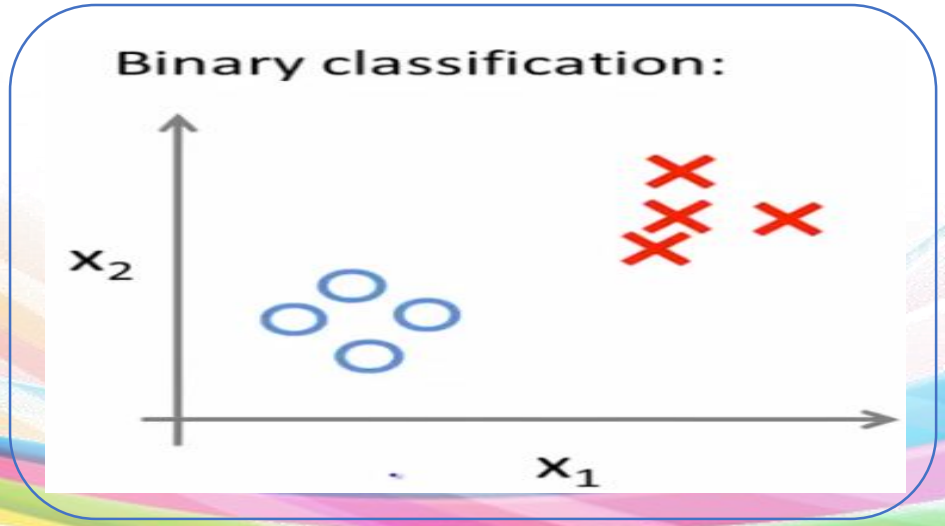
Advanced Optimization

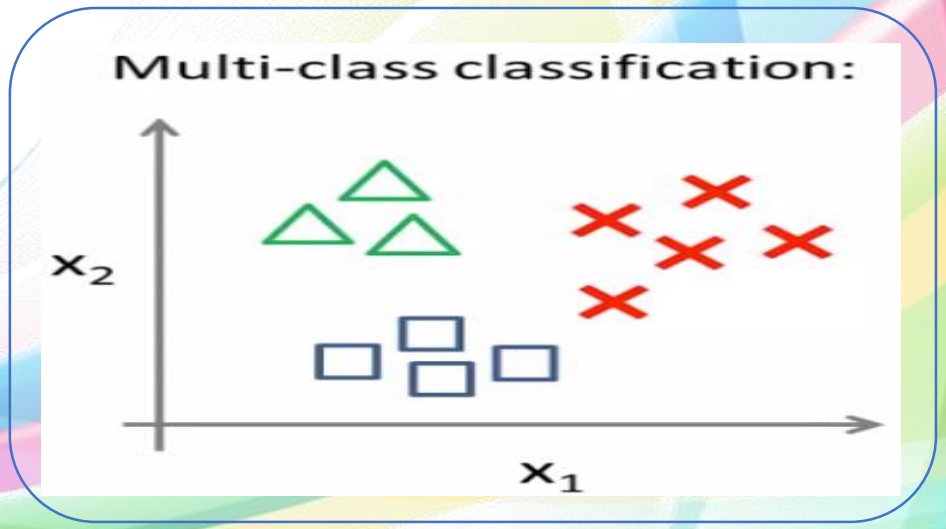
- To minimise cost, we need to calculate following two things:
  i. $J(\theta)$
  ii. $\frac{\partial}{\partial \theta_j} J(\theta)$
- After calculating above two things (actually only ii., i. is required when you want to monitor if cost is decreasing or not), Gradient Descent use these values to update the parameters, θs.
- However, there are many optimization algorithms like Gradient Descent, such as:
  1. Gradient Descent
  2. Conjugate Gradient
  3. BFGS
  4. L – BFGS
- The details of last three algorithms, i.e., Conjugate Gradient, BFGS and L – BFGS is beyond the scope of this course.
- Following are some of the pros and cons of these three algorithms, i.e., Conjugate Gradient, BFGS and L – BFGS:
  1. No need to manually pick learning rate (α)
  2. Often faster than Gradient Descent
  3. Very complex
- It is possible to use these algorithms successfully without even knowing there internal working.
- It is not advisable to implement these algorithms unless you are an expert of Numerical Computing.

Multiclass Classification

Multiclass Classification: One-vs-All

- Following are the examples of multiclass classification:
  1. Email foldering / tagging: Work, Friends, Family, Hobby
  2. Medical Diagrams: Not ill, Cold, Flu
  3. Weather: Sunny, Cloudy, Rainy, Snow
- Following is the depiction of difference between Binary and Multiclass Classification:

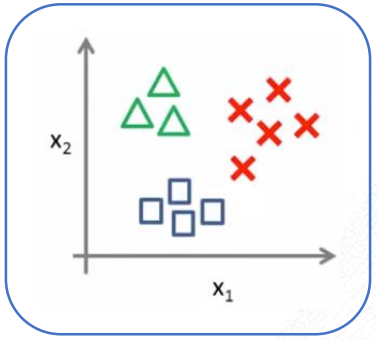




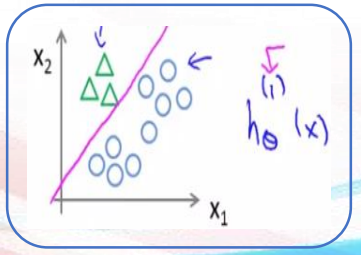
Multiclass Classification

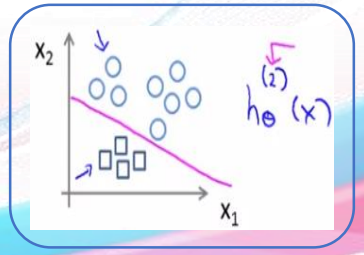
Multiclass Classification: One-vs-All

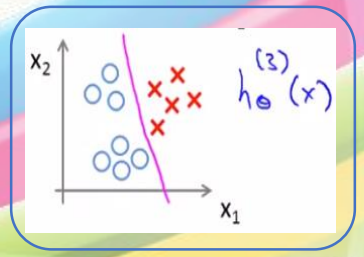
- Suppose, we have a data set with three classes: Triangle, Square, Cross. Following is how it looks:



- In One-vs-All technique, we will train three Logistic Regression classifier for each class as "1" and other classes as "0":







- Above we got three Logistic Regression models: $h^1(x)$, $h^2(x)$, $h^3(x)$. To make prediction, we will consider the maximum of these three models, i.e., prediction(x) = $\max(h^i(x))$, $i \in \{1, 2, 3\}$

Solving the Problem of Overfitting

The Problem of Overfitting

- So far, we have seen Linear Regression and Logistic Regression.
- These algorithms work well, however in some cases they run in a problem called as Overfitting.
- Overfitting can cause these ML algorithms to perform very poorly.
- Figure 1:
  - There is a plot of Linear Regression model prediction for housing price data with size of house on X-axis and price of house on Y-axis.
  - From the plot, it is clear that the model is not perfectly fit over the data.
  - This situation is called as Underfitting or High Bias.
  - High Bias because the model has pre-conception or biased towards linearity of data.
- Figure 2:
  - We have used a quadratic function in Linear Regression for housing price data.
  - From the plot, it is clear that it has worked pretty well.
- Figure 3:
  - We have used a complicated function in Linear Regression for housing price data.
  - From the plot, it seems that the function is doing well on training data set.
  - However, it is going quite up & down, thus it doesn't seem to be a good general fit.
  - This situation is called as Overfitting or High Variance.
  - High Variance because the hypothesis is too strict & fit only training data perfectly.

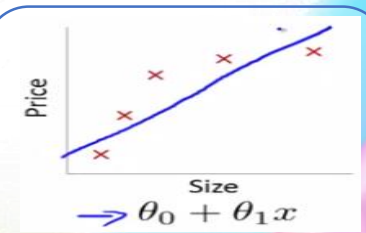


Figure 1

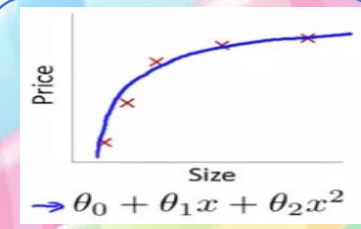


Figure 2

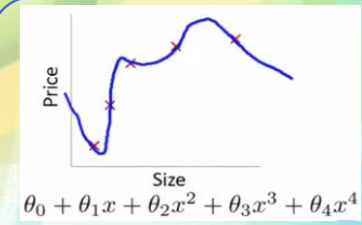
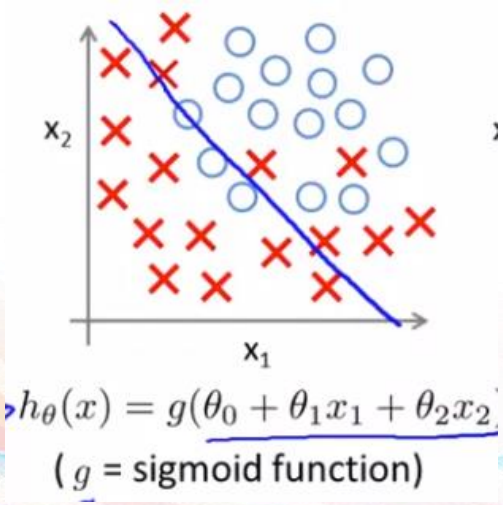


Figure 3

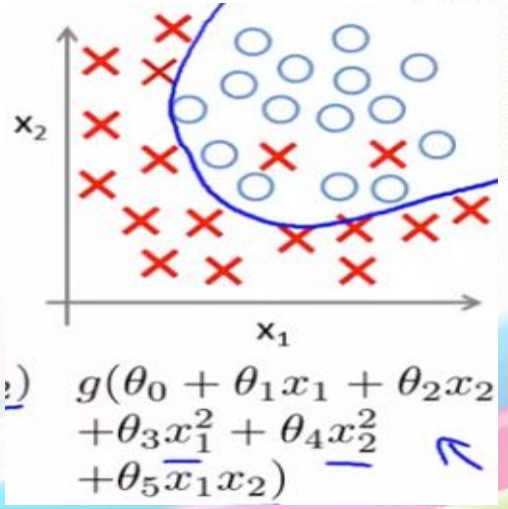
Solving the Problem of Overfitting

The Problem of Overfitting

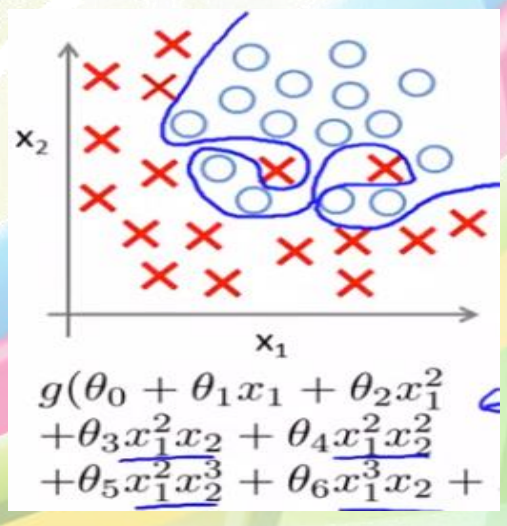
- Overfitting: If we have too many features, the learned hypothesis may fit the training set very well, but fail to generalize to new examples.
- Similar to Linear Regression (as we have seen in previous slid), Logistic Regression also suffer from Overfitting.
- Following are some examples of Logistic Regression suffering from different scenarios:



- Here, the Logistic Regression hypothesis function is simple.
- Hence, Underfitting / High Bias.



- Here, the hypothesis function is good and it seems a perfect fit.



- Here, the hypothesis function is very strict.
- Hence,Overfitting/High Variance.

Solving the Problem of Overfitting

The Problem of Overfitting

- Overfitting is a prevalent problem when we have lot of features and very less data.
- Here, we have two options to deal with this:
  1. Reduce number of features:
     i. Manually select which features to keep
     ii. Model Selection Algorithm: These algorithms tell which features are important and which are not.
  2. Regularization:
     - Keep all the features, but reduce magnitude / values of parameters, θ_j.
     - It works well when we have a lot of features, each of which contributes a bit to predicting y.

Solving the Problem of Overfitting

Cost Function

- We have seen that the plot of hypothesis function shown in Figure 1 is quite good.
- However, the plot of another hypothesis function which is quite complicated, shown in Figure 2 is strict and depicting the Overfitting or High Variance.
- To resolve this Overfitting or High Variance shown in Figure 2, we can penalize θ_3 & θ_4.

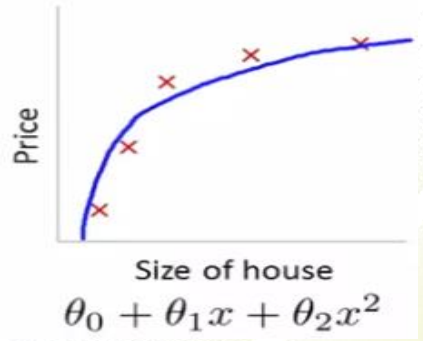


Figure 1

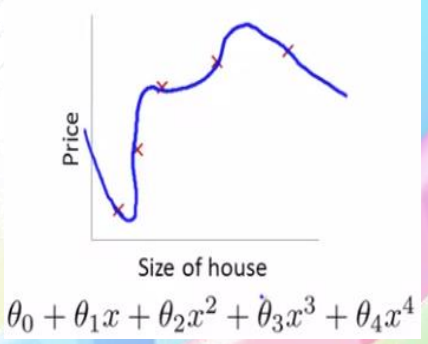


Figure 2

- Following is the method to penalize θ_3 & θ_4:

$$J(\theta) = \underbrace{\frac{1}{2m}\sum_{i=1}^{m}(h(x^i) - y^i)^2}_{\text{Quadratic Function}} + \underbrace{1000\,\theta_3^2 + 1000\,\theta_4^2}_{\text{Penalty}}$$

The only way to make above cost function small is by having very small values in θ_3 and θ_4. Thus, we will end up having $\theta_3 \approx 0$ and $\theta_4 \approx 0$.

Thus, we will left with a quadratic function and a Penalty with very little contribution. Consequently, we will get a perfect plot because of quadratic function (Figure 1) with little more flexibility or generalisability because of penalty term (Figure 2), that will eventually give us Figure 3 (Purple line).

Figure 3

Solving the Problem of Overfitting

Cost Function

- So, Regularization is actually about small values for parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_n$. This will give us:
  - Simpler hypothesis
  - Less prone to Overfitting
- Accepted by Andrew Ng that:

  "I realise that the reasoning of having all small value parameters may not be entirely clear to you know as it is hard to explain unless you implement it by yourself and see it by yourself."
- Now, in case of house pricing data set, we may have hundred features, like number of bedrooms, area, floor, etc., represented by $\{x_1, x_2, \ldots, x_{100}\}$. For these hundred features, we will be having 101 parameters, such as $\{\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}\}$. Since we do not know that which of these features to be minimised, hence we will add a Regularisation term in the cost function J(θ) that will be controlling all the parameters, like this:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h(x^i) - y^i)^2 + \underbrace{\lambda\sum_{j=1}^{n}\theta_j^2}_{\text{Regularization Term}}\right]$$

  Where, m: total number of instances, n: total number of features, λ: regularization parameter

  By convention, we regularize only θ_1 to θ_n, we do not regularize θ_0.

  The regularization parameter (λ) controls the trade-off between our two objectives:

  1. We would like to fit the training data well.
  2. We would like to keep the parameters (θs) small which will keep hypothesis simple & avoid Overfitting.
- If the regularization parameter (λ) is a very large number then it will result in Underfitting. How to choose? [LATER].

Solving the Problem of Overfitting

Regularised Linear Regression

- Earlier, we saw that the Gradient Descent algorithm for Linear Regression without Regularization is like this:

  Repeat {

  $$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_j$$

  } where, j = 0, 1, 2, …., n; m: total number of instances, n: total number of features, α: learning rate

- Now, we have Regularisation term also, so our Gradient Descent algorithm for Linear Regression will get changed:

  Repeat {

  $$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_0$$

  $$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_j + \frac{\lambda}{m}\theta_j\right]$$

  $$j \in \{1, 2, 3, \ldots, n\}$$

  } where, m: total number of instances, n: total number of features, α: learning rate, λ: regularization parameter

  The term under square brackets is equivalent of $\frac{\partial}{\partial\theta_j}J(\theta)$, where J(θ) includes the regularization term.

  The above θ_j can also be written as:

  $$\theta_j := \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h(x^i) - y^i\right)x^i{}_j$$

  Where, $j \in \{1, 2, 3, \ldots, n\}$ and $\left(1 - \alpha\frac{\lambda}{m}\right) < 1$ (usually). Thus, $\left(1 - \alpha\frac{\lambda}{m}\right)$ will make θ_j smaller (or shrink it) and then there is a usual Gradient Descent term to adjust (increase or decrease) it.

Solving the Problem of Overfitting

Regularized Linear Regression

- Earlier, we found that the Normal Equation solution for Linear Regression without Regularization is:

$$\theta = (X^T X)^{-1} X^T y$$

- Now, the Normal Equation solution for Linear Regression with Regularization will be:

$$\theta = (X^T X + \lambda M_{(n+1 \times n+1)})^{-1} X^T y$$

Where, n: total number of features, M: a (n+1 x n+1) identity matrix with $M_{00} = 0$. That is, if n = 2, then M:

$$M = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Skipped

Non-invertibility (optional / advanced) for Normal Equation with Regularization

Solving the Problem of Overfitting

Regularised Logistic Regression

- Earlier, we saw that the cost function of Logistic Regression without Regularization is like this:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log h(x^i) + (1 - y^i) \log(1 - h(x^i))\right]$$

  Where, m: total number of instances

- The following equation is the cost function of Logistic Regression with Regularization:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log h(x^i) + (1 - y^i) \log(1 - h(x^i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

  Where, m: total number of instances, λ: regularization parameter

- The Gradient Descent algorithm for Logistic Regression without Regularization is:

  Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left[\left(h\left(x^i\right) - y^i\right).x_j^i\right]$$

  } where, m: total number of instances, α: learning rate

- The Gradient Descent algorithm for Logistic Regression with Regularization is:

  Repeat {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left[\left(h\left(x^i\right) - y^i\right).x_0^i\right]$$

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left[\left(h\left(x^i\right) - y^i\right).x_j^i\right] + \frac{\lambda}{m}\sum_{j=1}^{m}\theta_j\right]$$

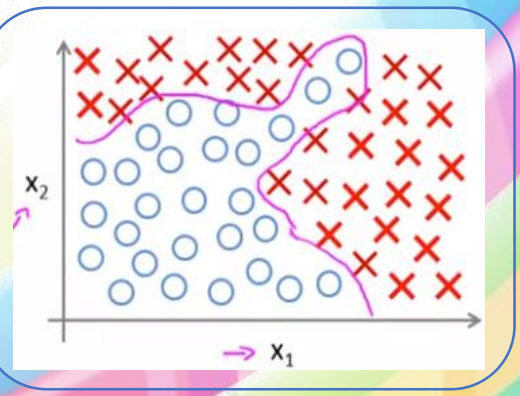
  } where, $j \in \{1, 2, 3, \ldots., n\}$, $h(x) = \frac{1}{1+e^{-\theta^T x}}$, m: total number of instances, α: learning rate

Motivations

Non-linear Hypotheses

- Neural Networks is a pretty old idea, but today it has become the state-of-the-art technique.
- In several Machine Learning problems, we need to have Machine Learning algorithms capable to learn complex non-linear hypotheses.
- In the corresponding figure, we have binary classification problem with two features, x_1 & x_2.
- Here, we can use Logistic Regression with a non-linear hypothesis function to fit well on this data, such as:

$$h(x) = g(\theta_0, \theta_1 x_1, \theta_2 x_2, \theta_3 x_1 x_2, \theta_4 x_1^2 x_2, \theta_5 x_1^3 x_2, \theta_6 x_1 x_2^2, \ldots..)$$



- Due to above complicated hypothesis function, we get a model represented by purple line.
- However, if we have many attributes, say 100, then Logistic Regression will become very cumbersome due to something like following terms in hypothesis function:

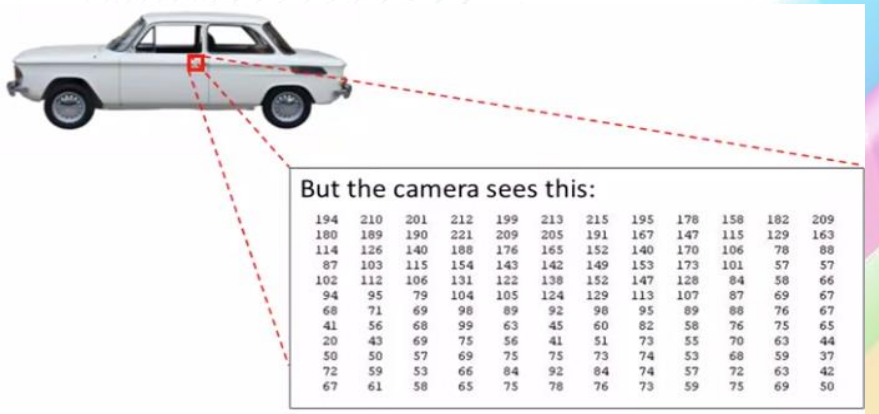
$$X_1^2, x_1x_2, x_1x_3, x_1x_4, \ldots., x_1x_{100},$$
$$X_2^2, x_2x_3, x_2x_4, x_2x_5, \ldots., x_2x_{100},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

This will be closed to 5,000 features. Thus, it may cause overfitting and also computationally expensive. Even if we take a subset of above features, we will may not get the perfect model with as much accuracy as shown in above figure.

Motivations

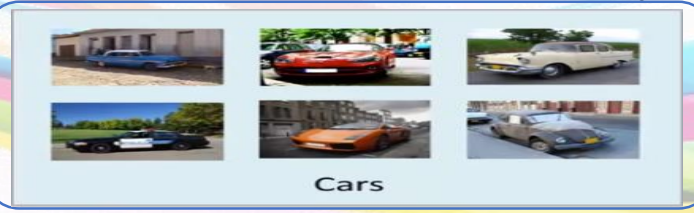
Non-linear Hypothesis

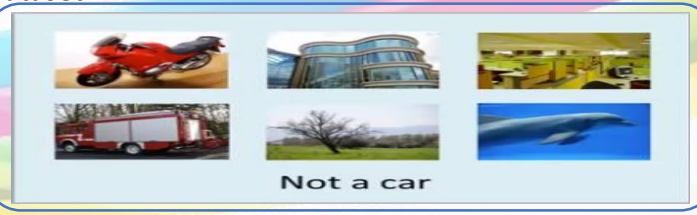
- For many Machine Learning problems, n (i.e., number of features) would be very high.
- The reason why Computer Vision problem is hard is that what we see in the given figure is a car but what a Computer Vision algorithm sees is a lot of numbers. Even for a tiny area of car, algorithm sees a lot of number as showing in the figure.



- Thus, feeding a whole image with lots of objects will provide huge number of features to our algorithm. In that case, algorithms like Logistic Regression will not be able to work.
- In a Computer Vision – Car Detection problem, we provide a data set of images with car and another without car and then we train our algorithm on that. After training, we test our trained model on an unseen image.

Training Dataset





Test Image – What is this?

Motivations

Non-linear Hypothesis

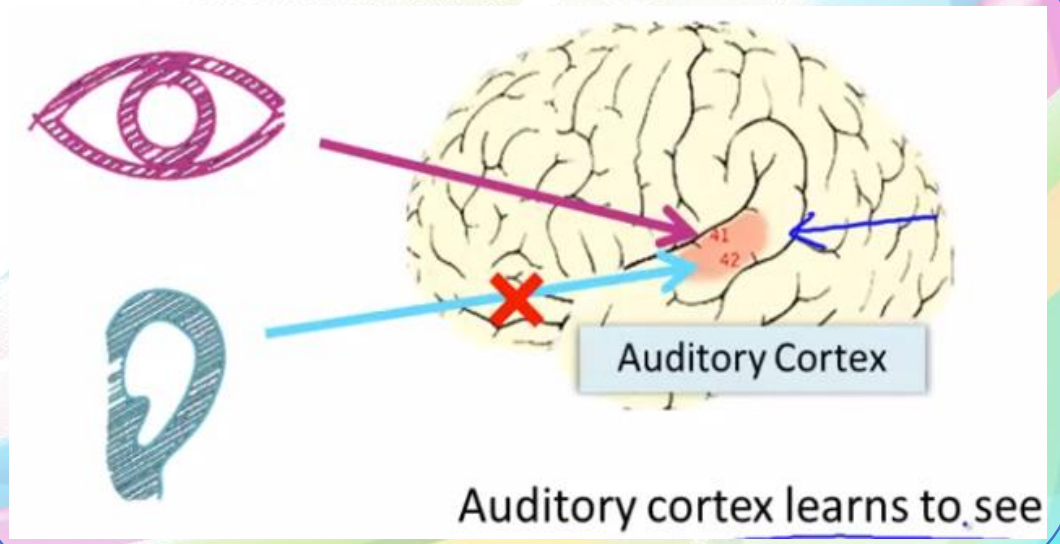
- Thus, for an extremely tiny grayscale image of size 50 x 50, we will have 2,500 pixels or features.
- Similarly, in case of three dimensions, these number of pixels or features will grow by three fold, i.e., 7,500.
- Each of these pixel values represent the brightness at that spot by a integer value in range 0 to 255.
- If we try to create all quadratic features for a 50 x 50 grayscale image to train on Logistic Regression, then we will get around 3 million features. Training Logistic Regression for these many features will be extremely computationally expensive.
- Similarly, for a 100 x 100 grayscale image, the total number of quadratic features will be around 50 million. Which is almost not possible to possible on a computer.

Motivations

Neurons and the Brain

- Neurons are inspired from Brain.
- Brain doesn't use several algorithms to learn different things. It has only one algorithm that it uses to learn a variety of things, such as observing, listening, speaking, etc.

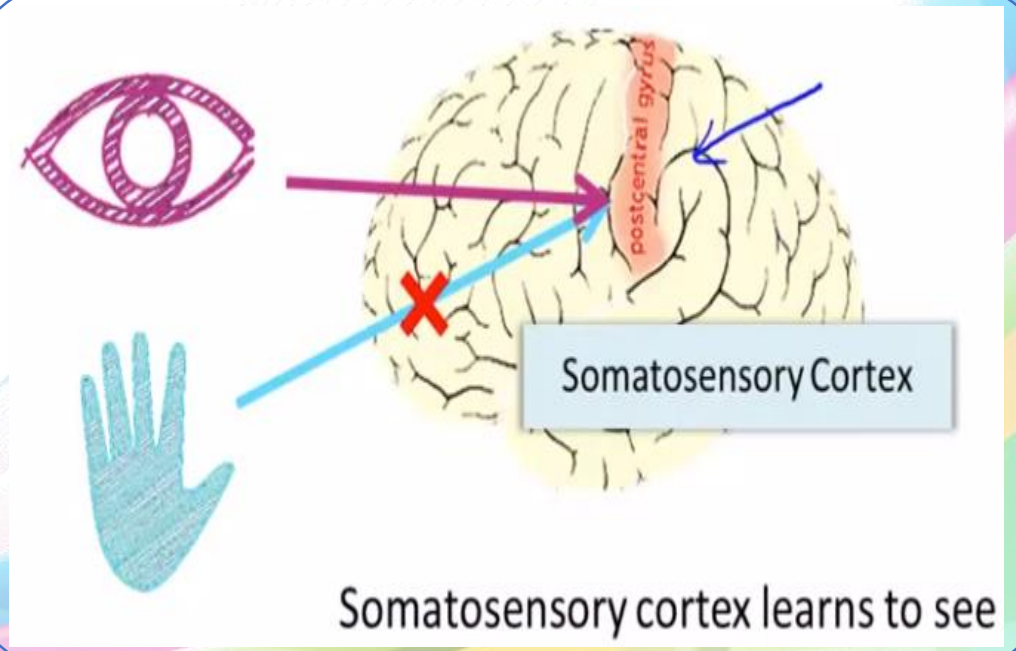
- Regarding this, an experiment was performed by scientists in which they cut the nerve to the part of brain responsible for listening, i.e., Auditory Cortex, and connected it with the nerve of eyes.
- Scientists found that the part of brain which was earlier doing listening, i.e., Auditory Cortex, is now learning observing.



Motivations

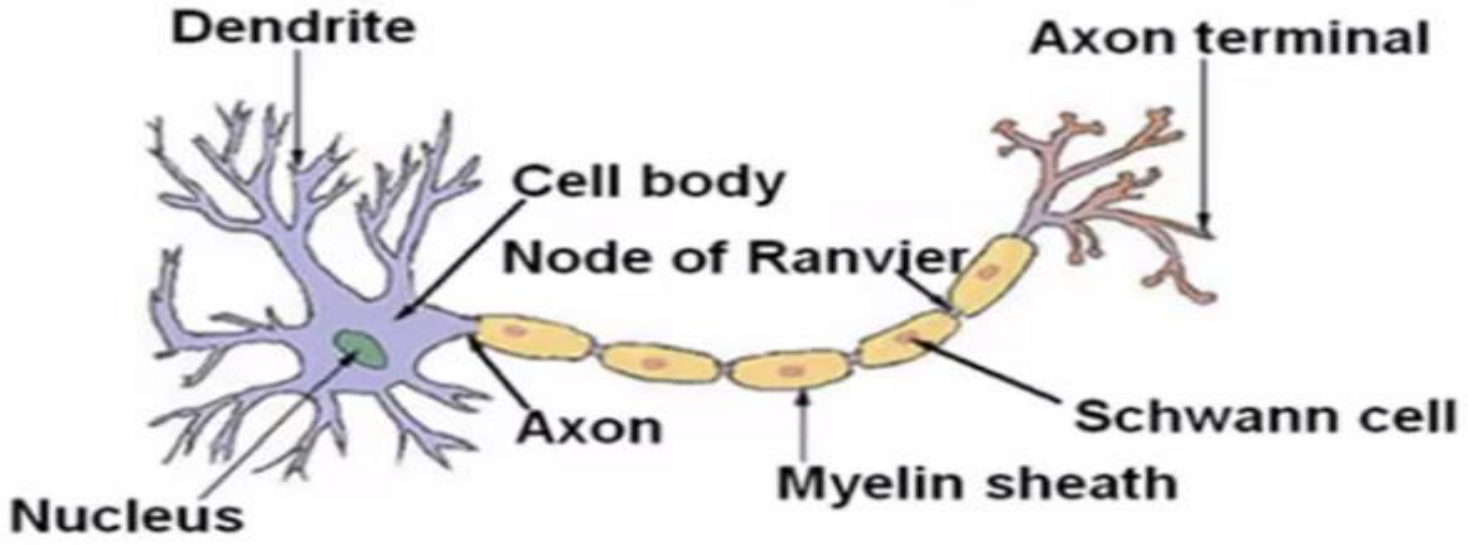
Neurons and the Brain

- Similarly, in another experiment, scientists connected the part of brain responsible for the senses of touch, i.e., Somatosensory Cortex, with the nerves of eyes.
- It was found that Somatosensory Cortex started learning to observe.
- These kind of experiments prove that our brain has only one kind of mechanism or algorithm to learn.
- Such experiments are called as Neuro Re-wiring experiments.



Neural Networks

Model Representation I

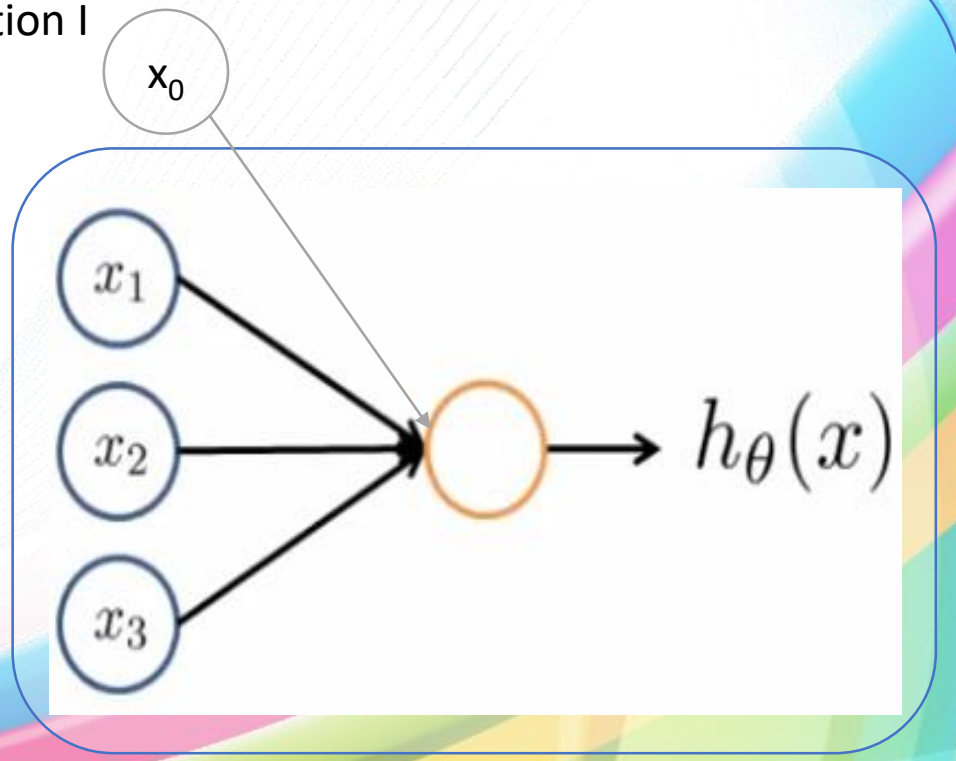


- Dendrite: They are inputs. They receive electric signals from other neurons for further processing.
- Axon: This is the output that passes the output signal, i.e., electric signals, to other neurons for further processing.
- Nucleus: This is the cell body that does processing of received information or electric signal.

Neural Networks

Model Representation I

- In an Artificial Neural Network (ANN), we have logistic units that perform the job of neurons.
- Here, incoming arrows to the yellow unit (or neuron) are inputs.
- The outgoing arrow from the yellow unit (or neuron) is the output.
- In the given figure, h(x) = $\frac{1}{(1+e^{-z})}$
- In an ANN, there will always be an addition unit, x_0, which is called as Bias Unit, and always equal to 1. Sometimes, it may or may not be represented, but you should always consider it irrespective of its presence or absence.



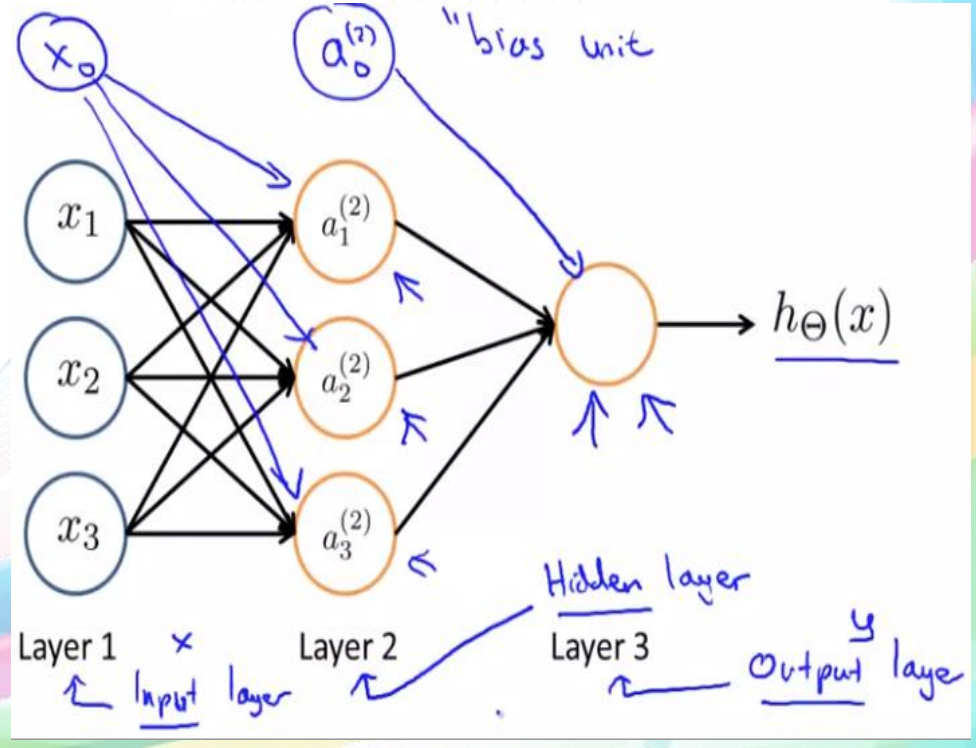
Neural Networks

Model Representation I

S.No.	Symbol	Definition
1	x_i	Input Node at Input Layer
2	a_i^j	Activation of unit "i" in layer "j"
3	z_i^j	Args. of activation of unit "i" in layer "j"
4	Θ^j	Weight mat. of mapping - layer j to (j + 1)

- Following are the details of activation function:
  $a_1^2 = g(\Theta_{10}^1 x_0 + \Theta_{11}^1 x_1 + \Theta_{12}^1 x_2 + \Theta_{13}^1 x_3) = g(z_1^2)$
  $a_2^2 = g(\Theta_{20}^1 x_0 + \Theta_{21}^1 x_1 + \Theta_{22}^1 x_2 + \Theta_{23}^1 x_3) = g(z_2^2)$
  $a_3^2 = g(\Theta_{30}^1 x_0 + \Theta_{31}^1 x_1 + \Theta_{32}^1 x_2 + \Theta_{33}^1 x_3) = g(z_3^2)$
  $h(x) = a_1^3 = g(\Theta_{10}^2 a_0^2 + \Theta_{11}^2 a_1^2 + \Theta_{12}^2 a_2^2 + \Theta_{13}^2 a_3^2)$
- If network has s_j units (excluding bias unit) in layer j, s_{j+1} units (excluding bias unit) in layer (j + 1), then Θ_j will be of dimension:

$$S_{j+1} \times (s_j + 1)$$



Neural Networks

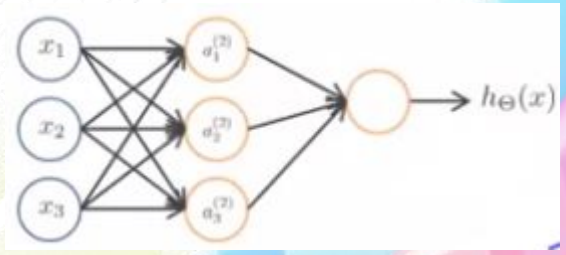
Model Representation II

- As we know that:

$$a_1^2 = g(\Theta_{10}^1 x_0 + \Theta_{11}^1 x_1 + \Theta_{12}^1 x_2 + \Theta_{13}^1 x_3) = g(z_1^2)$$
$$a_2^2 = g(\Theta_{20}^1 x_0 + \Theta_{21}^1 x_1 + \Theta_{22}^1 x_2 + \Theta_{23}^1 x_3) = g(z_2^2)$$
$$a_3^2 = g(\Theta_{30}^1 x_0 + \Theta_{31}^1 x_1 + \Theta_{32}^1 x_2 + \Theta_{33}^1 x_3) = g(z_3^2)$$

- The above equations can be represented in the form of vectors:

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad z^2 = \begin{pmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{pmatrix}$$



$z^2 = \Theta^1(x)$ (Θ^1 is the overall weight matrix of mapping from layer 1 to layer 2)

$a^2 = g(z^2)$ (a^2 is the overall activation of layer 2)

The above method of calculating activations from input layer, to hidden layers and ultimately to output layer, is called **Forward Propagation**.

- The way all the neurons and layers of a neural network are connected is called as Network Architecture.

Applications

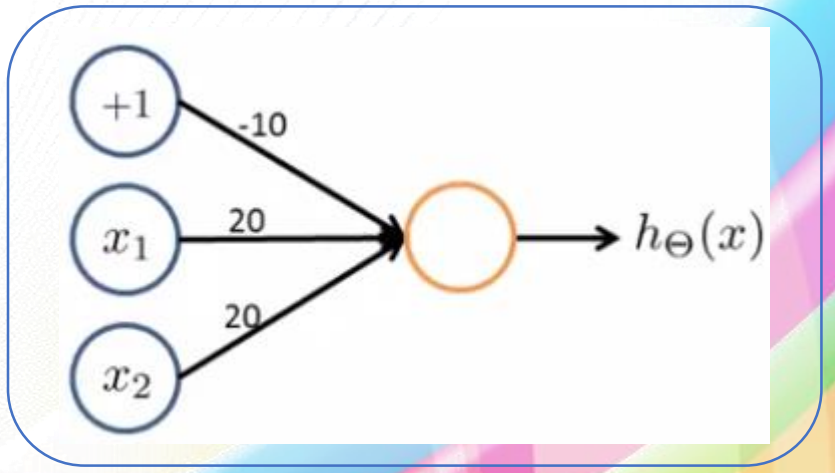
Examples and Intuitions I

- Given is a simple neural network.
- Suppose, input $(x_1, x_2) \in \{0, 1\}$
- This neural network is implementing one logic gate.
- Our job is to figure out the name of that logic gate.
- Here, we have following things:

  Inputs, x: $x_0 = 1$ $x_1 = \{0, 1\}$ $x_2 = \{0, 1\}$

  Weight, Θ: $\Theta_0 = -10$ $\Theta_1 = 20$ $\Theta_2 = 20$

  $h(x) = g(\Theta_0 x_0 + \Theta_1 x_1 + \Theta_2 x_2) = g(z) = \frac{1}{(1+e^{-z})}$



Now, we everything. Let us calculate and figure out the name of gate.

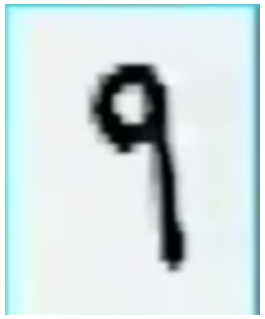
S.No.	x_0	x_1	X_2	z	g(z)	Output
1	1	0	0	-10x1 + 20x0 + 20x0 = -10	4.5×10^{-5}	Since g(z) < 0.5, hence 0
2	1	0	1	-10x1 + 20x0 + 20x1 = +10	0.999	Since g(z) > 0.5, hence 1
3	1	1	0	-10x1 + 20x1 + 20x0 = +10	0.999	Since g(z) > 0.5, hence 1
4	1	1	1	-10x1 + 20x1 + 20x1 = +30	1.0	Since g(z) > 0.5, hence 1

- Thus, this neural network is implementing OR gate.

Applications

Examples and Intuitions II

- Following is the visualisation of an input image after first few hidden layers of neural networks:

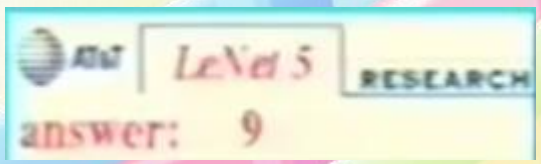


Input Image

Visualisation after 1st hidden layer

Visualisation after 2nd hidden layer

Visualisation after 3rd hidden layer



Output Image

Applications

Multiclass Classification: One vs All

- Now, suppose we have following four classes:

Pedestrian

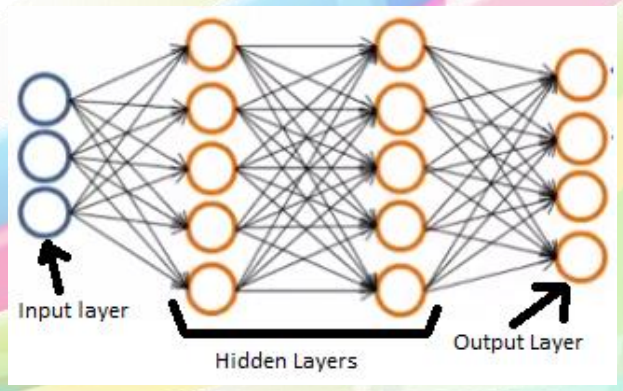
Car

Bike

Truck

- In the given multilayer ANN, we can see that there are as many units in the output layer as number of classes.
- Thus, unlike earlier, the output layer will give four outputs, like this:

For Pedestrian, $h(x) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ For Car, $h(x) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ For Bike, $h(x) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ For Truck, $h(x) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$



- The 1st, 2nd, 3rd, 4th unit in output layer is for respective classes.
- Due to this kind of architecture, our y will also have same format for each instance as described for h(x).

Cost Function and Backpropagation

Cost Function

- We will focus on the applications of Neural Networks (NN) in classification.
- Following are the representations of some of the symbols:

Symbol	Explanation
L	Total number of layers in the network
s_l	Number of units or neurons (not counting bias unit) in layer l

- In case of a Binary Classification, we will have only one output unit. Whereas, in case of a multiclass (K classes, K ≥ 3) classification, we will have K output units.
- The cost function in case of Logistic Regression looks like this:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log h(x^i) + (1 - y^i)\log(1 - h(x^i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- Now, the cost function of Neural Network is quite similar to above:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^i \log(h(x^{(i)}))_k + (1 - y_k^i)\log(1 - \log\left(h(x^i)\right)_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_{ji}^l)^2$$

h(x) is a K dimensional vector, where K is the total number of classes. $(h(x))_k$ (i.e., subscript k) denotes the k^{th} output of h(x).

The second term in the above equation is the regularisation term.

Cost Function and Backpropagation

Cost Function

- Suppose you want to try to minimise J(Θ) as a function of Θ, using one of the advanced optimization methods (fminunc, conjugate gradient, BFGS, L-BFGS, etc.). What do we need to supply code to compute (as a function of Θ)?

  Ans: J(Θ) and the partial derivative terms $\frac{\partial}{\partial \Theta_{ij}^{l}}$ for every i, j, l.

Cost Function and Backpropagation

Backpropagation Algorithm

- Backpropagation Algorithm is an algorithm to minimise the cost function in Neural Networks (NN).
- Following is the outline of Backpropagation Algorithm:

  Cost function of Neural Networks (NN):

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log(h(x^i))_k + (1 - y_k^i)\log(1 - \log\left(h(x^i)\right)_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_{ji}^l)^2$$

  Objective:

  Min $J(\Theta)$

  Need code to compute:

  i. $J(\Theta)$

  ii. $\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta)$

- Following is the vectorized implementation of Forward Propagation:

  $a^1 = x$

  $z^2 = \Theta^1 a^1$

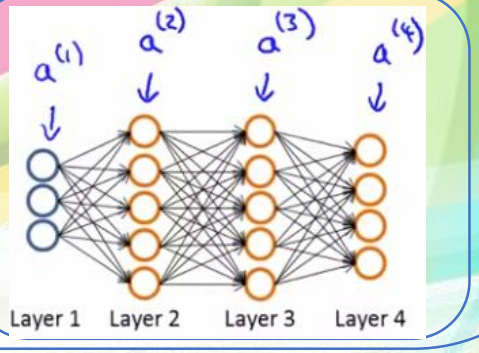
  $a^2 = g(z^2)$ (add a_0^2)

  $z^3 = \Theta^2 a^2$

  $a^3 = g(z^3)$ (add a_0^3)

  $z^4 = \Theta^3 a^3$

  $a^4 = h(x) = g(z^4)$



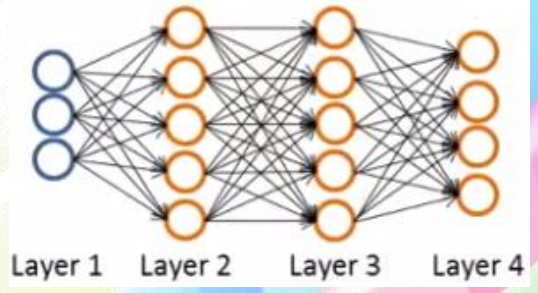
Cost Function and Backpropagation

Backpropagation

- In Backpropagation, we will calculate an error term, δ_j^l, for node j in layer l.
- In the given figure, we have a four layer artificial neural network where fourth layer is the output layer with four neurons or units, thus this artificial neural network can work for a four class classification problem.
- Thus, the error for each output unit j in layer 4 is:

$$\delta_j^4 = a_j^4 - y_j$$

$$\delta_j^4 = (h(x))_j - y_j$$



- Thus, in the vector form, we can write it for all units of a layer l as:

$$\delta^4 = a^4 - y$$

Similarly, error can be calculate for other earlier layers of neural network as:

$$\delta^3 = (\Theta^3)^T \delta^4 .* g'(z^3)$$

$$\delta^2 = (\Theta^2)^T \delta^3 .* g'(z^2)$$

Where, ".*" is the element wise multiplication operation of Matlab, "$g'(z^i)$" is "g-prime" which can be calculate as:

$$g'(z^i) = a^i .* (1 - a^i)$$

$g'(z^i)$ is actually the derivative of g function of the activation function.

There is no δ^1 because the first layer is the input layer that observes the features, that doesn't have any errors associated with it.

Cost Function and Backpropagation

Backpropagation

- The term Backpropagation comes from the fact that first the error is calculated for the final layer then the layer before it and so on.
- Thus, with advance maths, it can be prove that the partial derivative of cost function is like this:

$$\frac{\partial}{\partial \Theta_{ij}^{l}} J(\Theta) = a_j^l \, \delta_i^{l+1}$$

The above equation comes after ignoring regularization, i.e., λ or when λ = 0. We will fix this in later stages.

Thus, by performing backpropagation and computing delta terms (i.e., error), we can pretty quickly compute these partial derivative terms for all of your parameters.

- Backpropagation Algorithm:

Training Set $\{(x^1, y^1), (x^2, y^2), \ldots\ldots, (x^n, y^n)\}$

Set $\Delta_{ij}^l = 0$, (for all l, I, j; Δ is the capital of 'δ')

For i = 1 to m:

set $a^i = x^i$

Perform forward propagation to computer a^l for l = 1, 2, 3, …., L

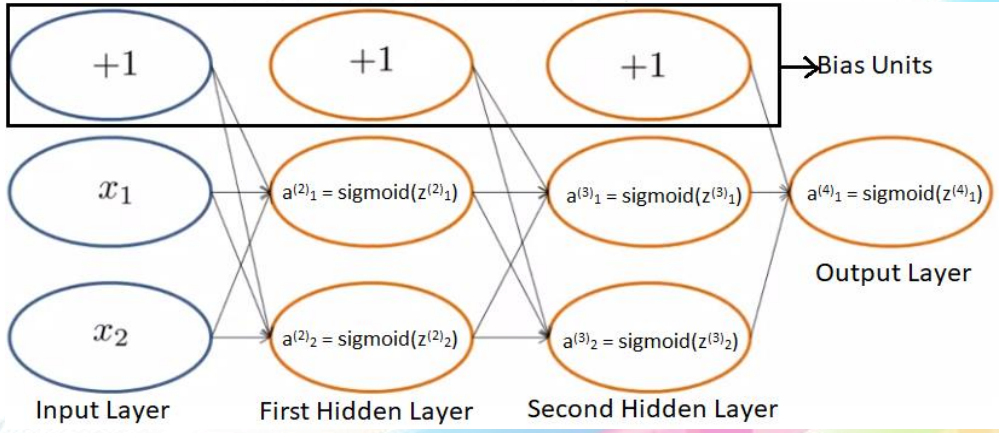
Using y^i, compute $\delta^{(L-1)}$, $\delta^{(L-2)}$, …., δ^2

$\Delta_{ij}^l := \Delta_{ij}^l + a_j^l \, \delta_i^{(l+1)}$

Cost Function and Backpropagation

Backpropagation Intuition

- To understand the intuition of Backpropagation, one has to first comprehend the concept of Forward Propagation.
- The given figure is depicting Forward Propagation.
- Following are the mathematical equations of Forward Propagation regarding this figure:



(x_1, x_2) are the input features of i^{th} instance of X.

$$a^{(2)}_1 = \text{sigmoid}(z^{(2)}_1) = \text{sigmoid}(\Theta^{(1)}_{10}(1) + \Theta^{(1)}_{11} x_1 + \Theta^{(1)}_{12} x_2)$$

$$a^{(2)}_2 = \text{sigmoid}(z^{(2)}_2) = \text{sigmoid}(\Theta^{(1)}_{20}(1) + \Theta^{(1)}_{21} x_1 + \Theta^{(1)}_{22} x_2)$$

$$a^{(3)}_1 = \text{sigmoid}(z^{(3)}_1) = \text{sigmoid}(\Theta^{(2)}_{10}(1) + \Theta^{(2)}_{11} a^{(2)}_1 + \Theta^{(2)}_{12} a^{(2)}_2)$$

$$a^{(3)}_2 = \text{sigmoid}(z^{(3)}_1) = \text{sigmoid}(\Theta^{(2)}_{20}(1) + \Theta^{(2)}_{21} a^{(2)}_1 + \Theta^{(2)}_{22} a^{(2)}_2)$$

$$a^{(4)}_1 = \text{sigmoid}(z^{(4)}_1) = \text{sigmoid}(\Theta^{(3)}_{10}(1) + \Theta^{(3)}_{11} a^{(3)}_1 + \Theta^{(3)}_{12} a^{(3)}_2)$$

- The computation of Forward propagation is explained above.
- Computations of Backward Propagation is quite similar to that of Forward Propagation but the only major difference is the direction of computation:

Forward Propagation →

← Backward Propagation

Cost Function and Backpropagation

Backpropagation Intuition

- Let us consider a simple neural network with only one output unit. The cost function in this case will look like this:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

- To understand Backpropagation, let us consider the above cost function equation for a single output unit but without regularisation (just for simplicity):

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h\left(x^{(i)}\right)\right)\right]$$

  Above equation can provide us cost for several instances (i ≥ 1). However, the cost of a i^{th} instance (a single instance) can be given by:

$$\text{cost}(i) = y^{(i)} \log(h(x^{(i)})) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

  The above equation place a row similar to the squared error. So, rather than considering the above cost for i^{th} instance, we can further simplify it as:

$$\text{cost}(i) \approx (h(x^{(i)}) - y^{(i)})^2$$

- What Backpropagation does, it calculates $\delta^{(l)}_j$ which is "error" of cost for $a^{(l)}_j$ (i.e., activation of unit j in layer l). Formally:

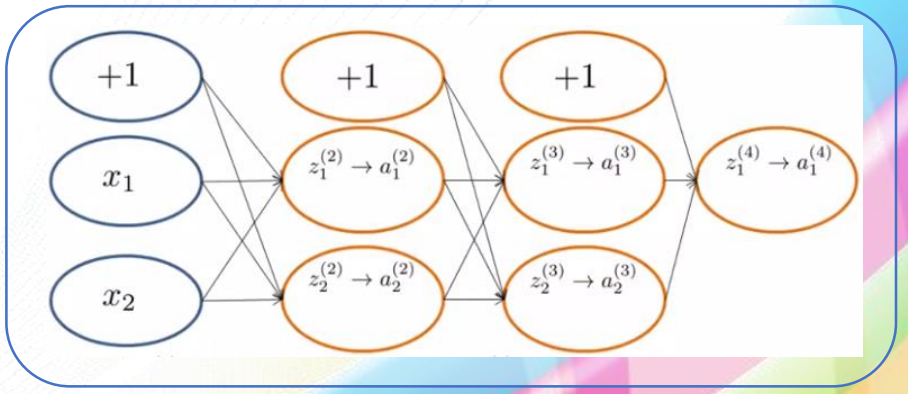
$$\delta^{(l)}_j = \frac{\partial}{\partial z_j^l}\text{cost}(i) \qquad (\text{for } j \geq 0)$$

Where, $\text{cost}(i) = \text{cost}(i) = y^{(i)} \log(h(x^{(i)})) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$

Cost Function and Backpropagation

Backpropagation Intuition

- Let us simplify it further and discuss without talking about partial derivatives. Consider the following neural network.



- For this neural network, the Backpropagation will calculate the error at unit of output layer for the i^{th} instance:

$$\delta^{(4)}_1 = y^{(i)} - a^{(4)}_1$$

- Next, the Backpropagation will propagate this error (i.e., $\delta^{(4)}_1$) to previous layers and calculate the same error at those layers, i.e., $\delta^{(3)}_1$, $\delta^{(3)}_2$, $\delta^{(2)}_1$ & $\delta^{(2)}_2$.
- These errors are calculated as follows:

$$\delta^{(2)}_2 = \Theta^{(2)}_{12}\, \delta^{(3)}_1 + \Theta^{(2)}_{22}\, \delta^{(3)}_2$$

$$\delta^{(3)}_2 = \Theta^{(3)}_{12}\, \delta^{(4)}_1$$

- Depending on your implementation of Backpropagation, you may calculate some error for Bias units but Bias units always output 1, so there is no way to change it. However, the way Andrew Ng does is implementing Backpropagation by just discarding Backpropagation error terms for Bias units.

Backpropagation in Practice

Implementation Note: Unrolling Parameters

- Here, we will unroll our parameters which are in the form of matrices into vectors.
- In advanced optimisation:

  function [jVal, gradient] = costFunction(theta)

  where jVal is the cost value.

  optTheta = fminunc(@costFunction, initialTheta, options)

  Above are some advanced functions that take parameters in vector form.

  For a four layer neural network, we have following things in the form of matrices:

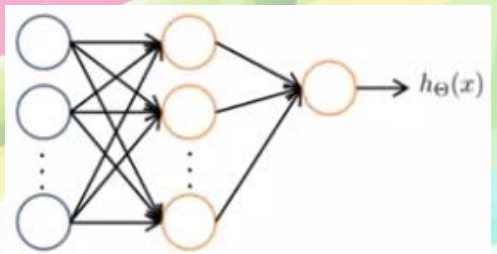
  $\Theta^{(1)}$, $\Theta^{(2)}$, $\Theta^{(3)}$ – weight matrices

  $D^{(1)}$, $D^{(2)}$, $D^{(3)}$ – gradient matrices

  We would like to convert them into vectors so that we can pass them to advanced optimisation functions.
- Now, suppose we have a neural network with 3 layers, and s_1, s_2 & s_3 units in them, where $s_1 = 10$, $s_2 = 10$ & $s_3 = 1$.
- The weights of this 3-layered neural network (shown in figure) for first, second and third layer are $\Theta^{(1)} \in R^{10x11}$, $\Theta^{(2)} \in R^{10x11}$ & $\Theta^{(3)} \in R^{1x11}$. Similarly, the gradients of first, second and third layer are $D^{(1)} \in R^{10x11}$, $D^{(2} \in R^{10x11)}$ & $D^{(3)} \in R^{1x11}$.
- In Octave, to convert these matrices into vectors, write this code:

  thetaVec = [Theta1(:); Theta2(:); Theta3(:)];

  Dvec = [D1(:); D2(:); D3(:)];



Backpropagation in Practice

Implementation Note: Unrolling Parameters

- Now, if you have weights (i.e., Θs) and gradients (i.e., Ds) in the form of vectors, then you can convert them into matrix by following piece of code:

Theta1 = reshape(thetaVec(1:110), 10, 11)	as $\Theta^{(1)} \in R^{10x11}$, 10 x 11 = 110 elements
Theta1 = reshape(thetaVec(111:220), 10, 11)	as $\Theta^{(2)} \in R^{10x11}$, 10 x 11 = 110 elements
Theta1 = reshape(thetaVec(221:231), 1, 11)	as $\Theta^{(3)} \in R^{10x11}$, 1 x 11 = 11 elements

Backpropagation in Practice

Gradient Checking

- Backpropagation is a tricky algorithm. It may be possible that after implementing Backpropagation, your cost would be reducing but there could be some bug in your implementation that will drastically impact the overall performance of your neural network.
- To resolve this, we have an idea or concept called as “Gradient Checking” that resolves all the issues regarding the implementation of Backpropagation.
- It is highly recommended that you always implement Gradient Checking as it will give you confidence about the implementation of your forward propagation and backpropagation, both.
- In the given figure, the value of cost function J(θ) at θ is shown.

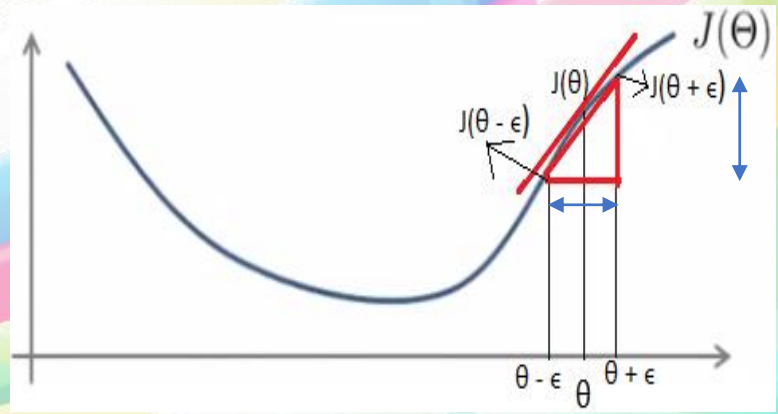
  The true derivative of J(θ) is the slope of tangent passing through J(θ), tangent shown in red color. We want to approximate this true derivative.

  To do this, we have taken a small value ϵ and find the value of J(θ + ϵ) at (θ + ϵ) and J(θ - ϵ) at (θ - ϵ).

  Now, connect J(θ - ϵ) and J(θ + ϵ) by a line and find out the derivative of this line. This can be done by dividing the perpendicular distance between J(θ - ϵ) and J(θ + ϵ) which is (J(θ + ϵ) - J(θ - ϵ)), by the horizontal distance between J(θ - ϵ) and J(θ + ϵ) which is 2ϵ.

  Thus, the approximate derivative of J(θ) can be given by:

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$



Backpropagation in Practice

Gradient Checking

- There is also an alternative formula to estimate the true derivative value of J(θ):

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{\epsilon}$$

In the above formula, there is only ϵ in the denominator, instead of 2ϵ as the earlier formula.

The formula with 2ϵ in the denominator is called as "two-sided difference" and the formula with only ϵ in the denominator is called as "one-sided difference".

Two-sided difference is more accurate than one-sided difference, thus two-sided difference is always preferred by Andrew Ng.

The value of ϵ should be very small, around 10^{-4} as it gives more accurate estimation of true derivative of J(θ). The value of ϵ should not be extremely small, otherwise you may run into mathematical issues because of too small value in denominator.

Following is the Octave code for this approximation:

$$\text{gradApprox} = \frac{J(theta+EPSILOn)-J(theta-EPSILON)}{(2*EPSILON)}$$

Backpropagation in Practice

Gradient Checking

- As we have $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$. The approximation for the derivatives w.r.t each of them, i.e., $\theta_1, \theta_2, \ldots, \theta_n$ would be:

$$\frac{\partial}{\partial \boldsymbol{\theta_1}} J(\theta) \approx \frac{J(\boldsymbol{\theta_1} + \epsilon, \theta_2, \ldots, \theta_n) - J(\boldsymbol{\theta_1} - \epsilon, \theta_2, \ldots, \theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \boldsymbol{\theta_2}} J(\theta) \approx \frac{J(\theta_1, \boldsymbol{\theta_2} + \epsilon, \ldots, \theta_n) - J(\theta_1, \boldsymbol{\theta_2} - \epsilon, \ldots, \theta_n)}{2\epsilon}$$

.

.

$$\frac{\partial}{\partial \boldsymbol{\theta_n}} J(\theta) \approx \frac{J(\theta_1, \theta_2, \ldots, \boldsymbol{\theta_n} + \epsilon) - J(\theta_1, \theta_2, \ldots, \boldsymbol{\theta_n} - \epsilon)}{2\epsilon}$$

- Above equations will help you to numerically approximate the value of J (i.e., cost function) for any of your parameters (i.e., $\theta_1, \theta_2, \theta_3, \ldots\ldots\ldots, \theta_n$).
- Following is the Octave implementation of the above equations:

```
for i = 1:n
        thetaPlus = theta;
        thetaPlus(i) = thetaPlus(i) + EPSILON;
        thetaMinus = theta;
        thetaMinus = thetaMinus(i) – EPSILON;
        gradApprox = (J(thetaPlus) – J(thetaMinus)) / (2 * EPSILON);
end;
```

Backpropagation in Practice

Gradient Checking

- So, earlier we have calculated the gradApprox which is the estimated value of our gradient. Now, DVec is the value of gradients that we will get from Backpropagation. Hence, we have to verify the following for Gradient Checking:

  $$\text{gradApprox} \approx \text{DVec}$$

  So, if the value of gradApprox and DVec is almost equal up to some decimal points then we could be confident about the implementation of Backpropagation and calculation of gradients.
- Following are the crucial points regarding Backpropagation and Gradient Checking:

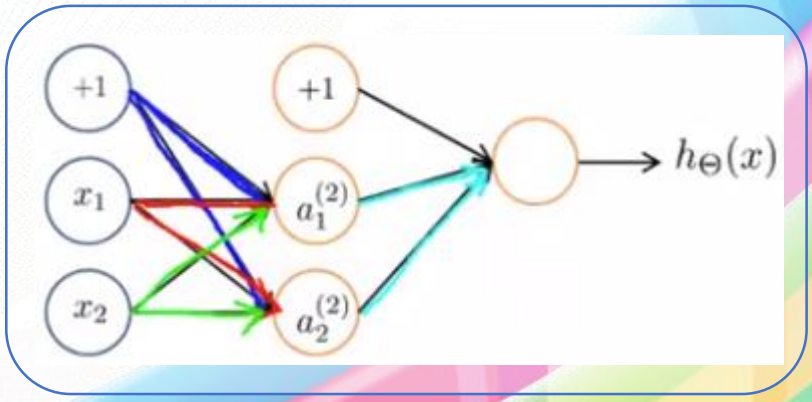
  Implementation Notes:
  - Implement Backpropagation to compute DVec (unrolled $D^{(1)}$, $D^{(2)}$, $D^{(3)}$).
  - Implement numerical gradient check to compute gradApprox.
  - Make sure they give similar values.
  - Turn off gradient checking while using Backpropagation for training.

  Important:
  - Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent then your code will be extremely slow.
- The main reason of using Backpropagation (despite its hard implementation) while training instead of using numerical approximation of gradients (despite their easier implementation) is the time. Numerical approximation of gradient is extremely slower than Backpropagation, thus Backpropagation is preferred.

Backpropagation in Practice

Random Initialization

- For Gradient Descent and Advanced Optimization, we need initial values for Θ.
- Initialising theta (i.e., weight matrix) by zeros is fine for Logistic Regression. However, it doesn't work for Neural Networks.
- Consider the following figure. Here, $\Theta^{l}_{ij} = 0$, for all i, j and l, initially.
- When we have same weights for the first unit of first layer (marked with blue arrows), similarly for second unit of first layer (marked with red arrows) and again for third unit of first layer (marked with green arrows), we will get same activation values at second layer, i.e., $a^{(2)}_1 = a^{(2)}_2$.
- Moreover, the error calculated at the end will also be same, i.e., $\delta^{(2)}_1 = \delta^{(2)}_2$.



- Eventually, after each update, parameters corresponding to inputs going into each of two hidden units are identical. Thus,

$$\Theta^{(1)}_{01} = \Theta^{(1)}_{02} \rightarrow a^{(2)}_1 = a^{(2)}_2 \qquad \text{(As earlier)}$$

  As a result, your neural network will see only one feature at the end and it will prevent neural network from learning from data.
- This problem is known as Symmetric Weight (i.e., same weights).

Backpropagation in Practice

Random Initialization

- In order to resolve this problem (problem of Symmetric Weights), we use random initialization for our parameters that cause "symmetry breaking".
- Thus, initialize each $\Theta^{(l)}_{ij}$ to a random value in $[-\epsilon, +\epsilon]$, i.e., $-\epsilon \leq \Theta^{(l)}_{ij} \leq \epsilon$.
- Octave code to do the above initialization is:

  *Theta1 = rand(10, 11) * (2 * INIT_EPSILON) – INIT_EPSILON;*

  *Theta2 = rand(1, 11) * (2 * INIT_EPSILON) – INIT_EPSILON;*

  The function rand(x, y) will generate a matrix of dimension (x, y) with values between 0 and 1. Then, multiply the generated matrix with (2 * INIT_EPSILON) and finally, subtract it with INIT_EPSILON. This will make all values of Theta between $-\epsilon$ and $+\epsilon$.
- This ϵ has nothing to do with the ϵ that we were using in Gradient Checking.
- Consider this procedure for initializing the parameters of a neural network:
  1. Pick a random number r = rand(1, 1) * (2 * INIT_EPSILON) – INIT_EPSILON;
  2. Set $\Theta^{(l)}_{ij}$ = r, for all i, j, l.

  Does this work?

  Ans: No, because this fails to break symmetry as every element of $\Theta^{(l)}_{ij}$ will have same value. After every gradient descent update, all the values will get the same update and again remain same. Thus, it will not be able to break symmetry.

Backpropagation in Practice

Putting It Together

- Before training a neural network, you have to pick a network architecture, i.e., connectivity patterns between neurons.
- In your architecture, following things will always be well defined due to dependency on input data:
  # Number of input units: Dimension of features of input data. # Number of output units: Number of classes.
- Number of hidden layers:
  - Reasonable default is 1 hidden layer.
  - It can be more than 1 and can have the same number of hidden units in each layer (reasonable default).
  - Usually more the number of units in hidden layer is better. However, this will raise the computation.
  - Moreover, number of units in each hidden layer must be mathematically compatible with that of earlier layer.
- Training a Neural Network:
  1. Randomly initialize weights.
  2. Implement forward propagation to get $h(x^{(i)})$ for any $x^{(i)}$.
  3. Implement code to compute cost function J(Θ).
  4. Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
  5. Use gradient descent checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of J(Θ). Then, disable gradient checking code.
  6. Use gradient descent or advanced optimization method with backpropagation to try to minimize J(Θ) as a function of parameters Θ.

Backpropagation in Practice

Putting It Together

- Following is the pseudocode for Forward Propagation and Backpropagation:

  for i = 1:m

  Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

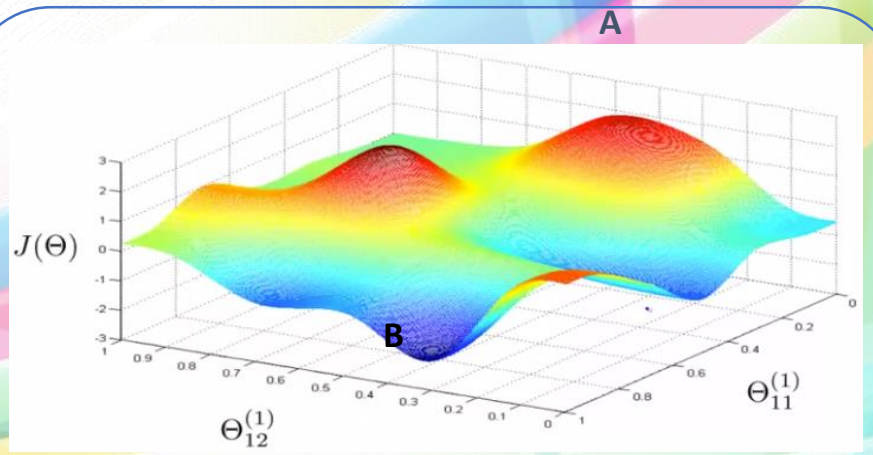
  (Get activation $a^{(l)}$ *and delta term* $\delta^{(l)}$ *for l = 2, 3, …., L)*

  $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

  Compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

- Usually, no one can guarantee that Gradient Descent will find the global minimum, but mostly it performs a reasonably good job by finding a good local minimum.
- Following is the figure of cost function J(Θ). In this figure, at point A, where cost is quite high, the value of $h(x^{(i)})$ will be far from $y^{(i)}$. Whereas, at point B, where cost is quite low, the value of $h(x^{(i)})$ will approximately equal to $y^{(i)}$.
- Suppose you are using gradient descent together with backpropagation to try to minimize J(Θ) as a function of Θ. A useful step to verify that the learning algorithm is running correctly:

  Plot J(Θ) as a function of the number of iterations and make sure it is decreasing (or at least non-increasing) with every iteration.



Backpropagation in Practice

Putting It Together

- Andrew Ng:

  "Neural Network learning and Backpropagation is a complicated algorithm. And even though I've seen the math behind back propagation for many years and I've used back propagation, I think very successfully, for many years, even today I still feel like I don't always have a great grasp of exactly what back propagation is doing sometimes."

- If we are training a neural network using gradient descent, one reasonable debugging step is to make sure it is working is to plot J(Θ) as a function of the number of iterations, and make sure it is decreasing (or at least non-increasing) after each iteration.

Evaluating a Learning Algorithm

Deciding What to Try Next

- Now, we will see how to effectively use Machine Learning algorithms with some practical suggestions given by Andrew Ng.
- Debugging a learning algorithm:

  Suppose you have implemented regularized linear regression to predict housing prices:

  $$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

  However, when you test your hypothesis on a new set of houses, you find that it makes unacceptable large errors in its predictions. What should you try next?

  Options:

  - Get more training examples
  - Try smaller sets of features
  - Try collecting additional features
  - Try adding polynomial features
  - Try decreasing λ
  - Try increasing λ

  Many people randomly selects any of above options, say collecting more data, gathering additional features, etc., and waste six months of their life in doing something that was either not required or not a good idea.
- There are pretty simple techniques to rule out half of the above options in order to search for the best option without wasting your time.

Evaluating a Learning Algorithm

Deciding What to Try Next

- These techniques are called as Machine Learning Diagnostics.
- Machine Learning Diagnostics: A test that you can run to gain insight what is / isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.
- Diagnostics can take time to implement, but dong so can be very good use of your time as they can save your many months.
- Summarising Machine Learning Diagnostics:
  - Diagnostics can give guidance as to what might be more fruitful things to try to improve a learning algorithm.
  - Diagnostics can be time consuming to implement and try, but they can still be a very good use of your time.
  - A diagnostics can sometimes rule out certain courses of action (changes to your learning algorithm) as being unlikely to improve its performance significantly.

Evaluating a Learning Algorithm

Evaluating a Hypothesis

- As we know that if a hypothesis has low training error, it doesn't mean that hypothesis or learning algorithm is good as they may fail to generalize to new instances not in training set.
- In case of one feature, we can plot the hypothesis and see how it looks like and its performance. However, in case of several features plotting hypothesis function would be impossible. Thus, we need some other way to evaluate the hypothesis.
- To evaluate our hypothesis, one can split the entire given data set into two parts:
  1. Training Set:
     - 70% of the entire data set.
     - Represented as: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots., (x^{(m)}, y^{(m)})\}$
  2. Test Set:
     - Remaining 30% of the entire data set.
     - Represented as: $\{(x^{(1)}_{test}, y^{(1)}_{test}), (x^{(2)}_{test}, y^{(2)}_{test}), \ldots., (x^{(mtest)}_{test}, y^{(mtest)}_{test})\}$
- If there is any kind of sorting over the data, then it would be better to randomly choose 70% of data for training and remaining randomly left 30% of data for testing.

Evaluating a Learning Algorithm

Evaluating a Hypothesis

- Training / testing procedure for Linear Regression:
  1. Learn parameter θ from training data (minimizing training error J(θ))
  2. Compute test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h\left(x_{test}^{(i)}\right) - y_{test}^{(i)})^2$$

- Training / testing procedure for Logistic Regression:
  1. Learn parameter θ from training data (minimizing training error J(θ))
  2. Compute test error:

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \log(h\left(x_{test}^{(i)}\right) + \left(1 - y_{test}^{(i)}\right) \log(h(x_{test}^{(i)})\right]$$

Well, the above test error term is quite reasonable but sometimes people use misclassification error (aka 0 / 1 misclassification error) which means either a test instance is predicted correctly (0 or 1) or incorrectly (1 or 0). Following is its formula:

Misclassification Error (or 0/1 misclassification error) = $\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h\left(x_{test}^{(i)}\right), y_{test}^{(i)})$

Evaluating a Learning Algorithm

Model Selection and Train / Validation / Test Sets

- Suppose you wish to decide that what degree of polynomial to fit a data set, or what features to include for a learning algorithm, or to choose a regularization parameter (λ), how to do that.
  These are called as Model Selection Problems.
- Once parameters θ_0, θ_1, θ_2, …., θ_n were fit to some set of data (training set), the error of the parameters as measured on that data (the training error (J(θ)) is likely to be lower than the actual generalization error.
- Due to this, the training error is not a good evaluator for deciding that how well the hypothesis will perform on the new data set (i.e., the test set).
- Suppose, you have 10 choices for your hypothesis functions:
  1. $h(x) = \theta_0 + \theta_1 x$ (a linear equation)
  2. $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ (a quadratic equation)
  3. $h(x) = \theta_0 + \theta_1 x + \ldots + \theta_3 x^3$ (a cubic equation)
  .
  .
  10. $h(x) = \theta_0 + \theta_1 x + \ldots. + \theta_{10} x^{10}$ (a 10-degree polynomial)

  So, we have one more parameter d, denoting the degree of polynomial equation, i.e., d = 1 for linear equation, d = 2 for quadratic equation, …., d = 10 for 10-degree polynomial equation.

Evaluating a Learning Algorithm

Model Selection and Train / Validation / Test Sets

Now, assume each of our previous hypothesis gave us a set of parameters represented by $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(10)}\}$. Then, we calculated the respective cost of each of the earlier hypothesis on test set, let's say they are $J_{test}(\theta^{(1)})$, $J_{test}(\theta^{(2)})$, $J_{test}(\theta^{(3)})$, ….., $J_{test}(\theta^{(10)})$.

Now, it seems logical that the hypothesis with the lowest error on test set should be chosen, however it will not help us to choose the best hypothesis that can generalize well on unseen data set.

- To address this problem, we can split our data set into three parts:
  1. Training Set: Usually 60% of the entire data set
  2. Cross Validation Set (or simply Validation Set): Usually 20% of the entire data set
  3. Test Set: Usually 20% of the entire data set

Now, we can calculate following three errors for each of our probable hypothesis:

1. Train Error: $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$
2. Cross Validation Error: $J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m}(h(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$
3. Test Error: $J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m}(h(x_{test}^{(i)}) - y_{test}^{(i)})^2$

Now, we will calculate $J_{cv}(\theta)$ for each of our hypothesis and choose the one with lowest cross-validation error. To further check it for generalisation error, we can use test set.

Evaluating a Learning Algorithm

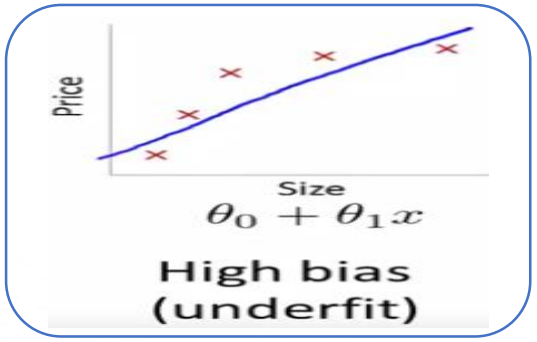
Model Selection and Train / Validation / Test Sets

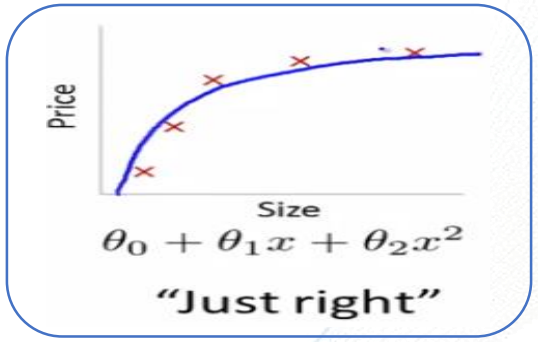
- Consider the model selection procedure where we choose the degree of polynomial using a cross-validation set. For the final model (with parameters θ), we might generally expect $J_{cv}(\theta)$ to be lower than $J_{test}(\theta)$ because:
  - An extra parameter (d, the degree of the polynomial) has been fit to the cross validation set for selecting the final model out of all probable models.

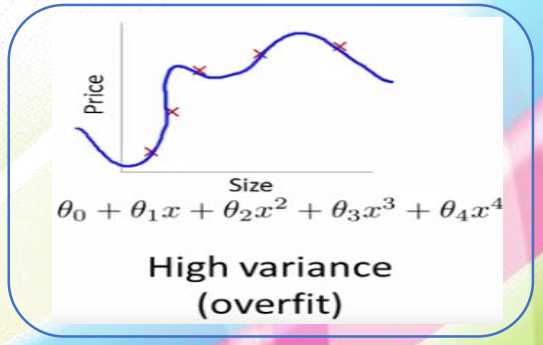
Bias vs. Variance

Diagnosing Bias vs. Variance

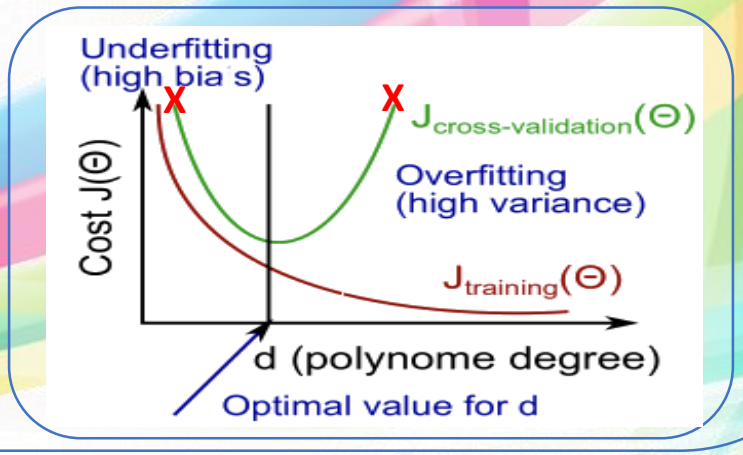
- Whenever you run a learning algorithm and you are not getting the expected result, then most of the time it is because of either high bias or high variance problem, in other words, underfitting or overfitting.







- The left peak indicates Bias and the right peak indicates Variance. Thus, by plotting the cross-validation and training error, one can figure out if his hypothesis function is suffering from Bias or Variance.
- In case of Bias (underfit):
  - $J_{train}(\theta)$ will be high
  - $J_{cv}(\theta) \approx J_{train}(\theta)$
- In case of Variance (overfit):
  - $J_{train}(\theta)$ will be low
  - $J_{cv}(\theta) >> J_{train}(\theta)$



Bias vs. Variance

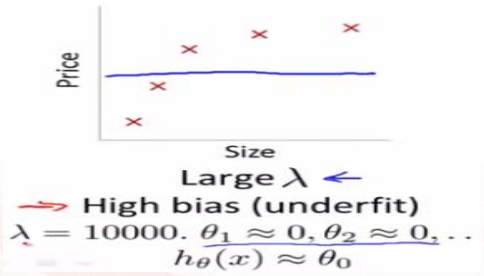
Regularization and Bias / Variance

- We know that how Regularisation can help preventing overfitting but how does it affect the bias and variance of a learning algorithm.
- Suppose we have a model or hypothesis function like shown below:

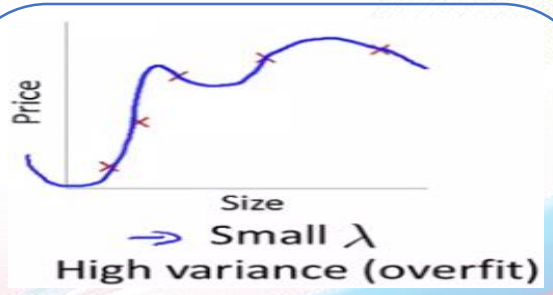
$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Its cost function can be given by: $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h(x^i) - y^i)^2 + \underbrace{\lambda \sum_{j=1}^{n} \theta_j^2}\right]$

Regularization Term



Since regularization parameter (λ) is high, hence parameters (θs) will be heavily penalize & become close to 0 resulting into high bias, i.e., underfitting.



When the regularization parameter (λ) is small (close to 0), learning parameters (θs) will not be penalize resulting into high variance, i.e., overfitting.

When regularization parameter (λ) is optimized, then the learning parameters (θs) will also be perfect and results into a right model.

Bias vs. Variance

Regularization and Bias / Variance

- Now, the question is how we get a optimized value of regularization parameter (λ).
- In order to choose the value of regularization parameter (λ), we can calculate cost for several different values of λ:

Model or Hypothesis function: $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Cost function: $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$

1. Try λ = 0
2. Try λ = 0.01
3. Try λ = 0.02
4. Try λ = 0.04

.

.

12. Try λ = 10.24

Calculate $J_{cv}(\theta)$ for each of these λs and select the one that gives lowest $J_{cv}(\theta)$.

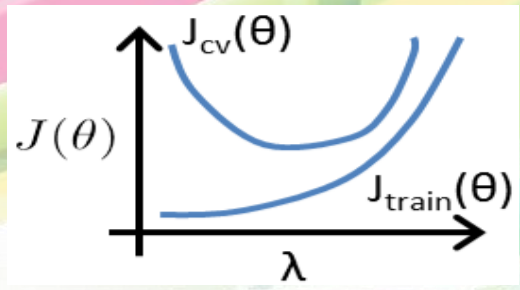
- Consider regularized logistic regression. Let:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m}(h(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

The plot of J_{train} and J_{cv} against λ will be as following:



Bias vs. Variance

Learning Curve

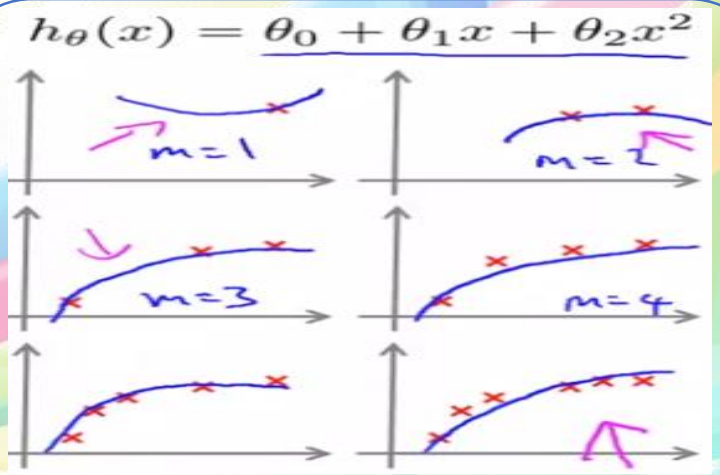
- Learning Curve is useful for following two things:
  - To check if your algorithm is working correctly or not
  - To improve the performance of the algorithm
- It is also useful to diagnose if a particular learning algorithm is suffering from bias or variance.
- Suppose following is the squared error of our training and cross-validation set:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m}(h(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Consider the following figures:

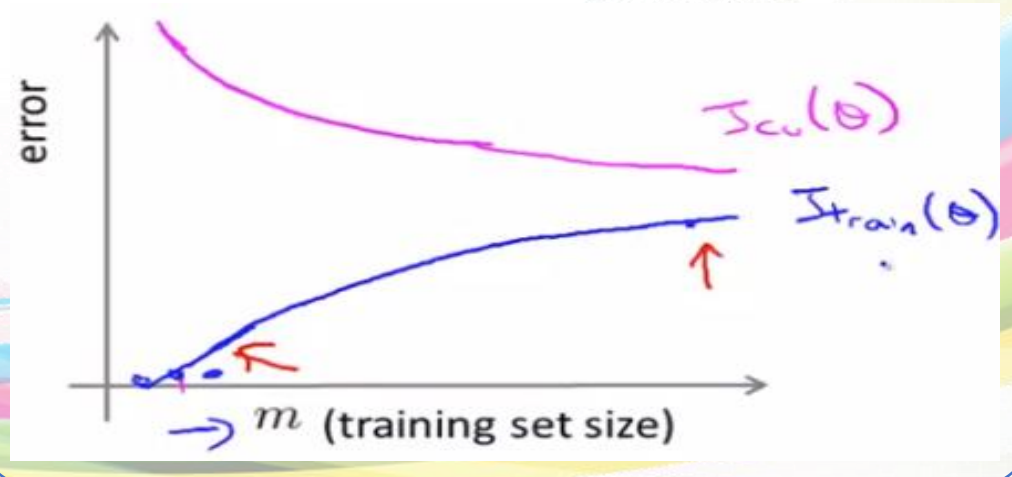
The given figure has the plots of hypothesis h(x) against the input feature x. From this figure, this is clear that as the number of instances m is increasing (represented by red cross mark), the performance of hypothesis is decreasing, i.e., there is more error.



Bias vs. Variance

Learning Curve

- Thus, from the previous slide it is clear that the training error increases as the number of instances in data set increases.
- Whereas, the cross-validation error decreases as the number of instances increases in data set because of more learning of learning algorithm.
- The above two points can be depicted in the following plot.
- This plot is between the error on Y-axis and m (i.e., number of instances in the data set) on X-axis.
- The blue line is the plot of $J_{train}(\theta)$ and pink line is the plot of $J_{cv}(\theta)$. This is the learning curve.



Bias vs. Variance

Learning Curve

- Now, we will see learning curve in case of high bias and high variance.
- Let us first consider the case of high bias:
  - To explain this, we will consider the equation of straight line on a non-linear data set: $h(x) = \theta_0 + \theta_1 x$
  - Now, our hypothesis (straight line model, given above) will look something like this – figure 1.
  - However, when we increase no. of data points in training set, we will get almost similar straight line – figure 2.
  - Thus, we can conclude that when training error will increase up to some point and then it becomes constant and validation error decrease up to some point and become constant, in case of high bias – see figure 3.
  - Since we have so few parameters and so much data, thus the performance of model on training and validation will become almost similar and results into high training and validation error.

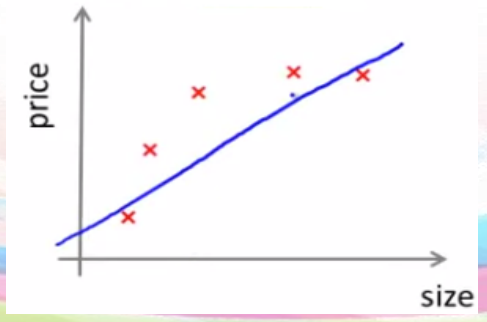


Figure 1

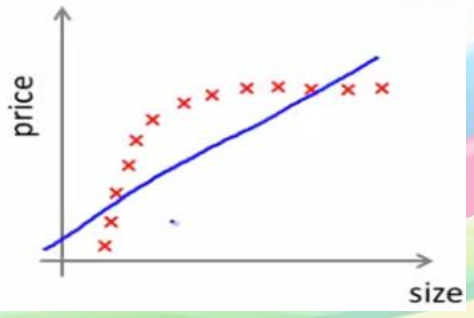


Figure 2

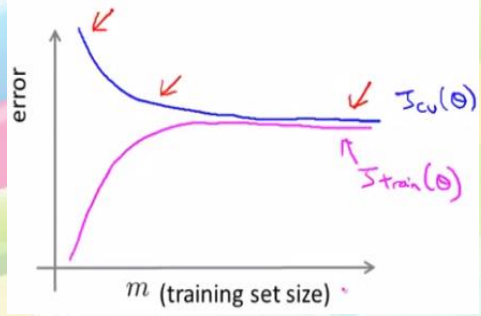


Figure 3

Bias vs. Variance

Learning Curve

- Thus from last slide, we can conclude that:

> If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

- Now, we will consider the case of high variance:
  - To describe this, suppose we have a 100^{th} degree polynomial hypothesis function with a small value of λ:

$$h(x) = \theta_0 + \theta_1 x + \ldots. + \theta_{100} x^{100}$$

  - For a small number of data points, we will end up fitting hypothesis very well – see figure 1.
  - Now, when training data set size becomes large, still out hypothesis will fit quite well but not as much perfectly as in the last point – see figure 2.
  - Thus, training error will increase with increase in data size and validation error will decrease up to a point, however there will be a huge gap between. Validation error will be quite high due to overfitting.

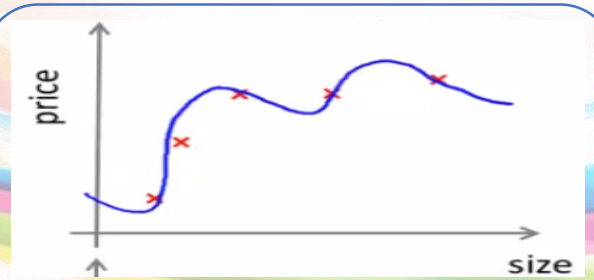


Figure 1

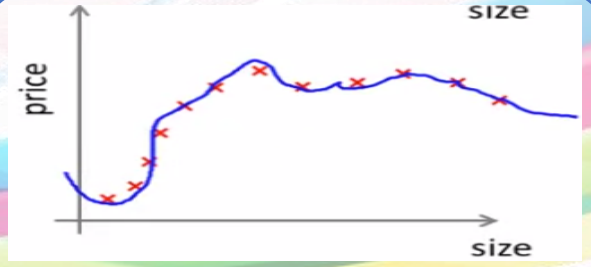


Figure 2

Figure 3

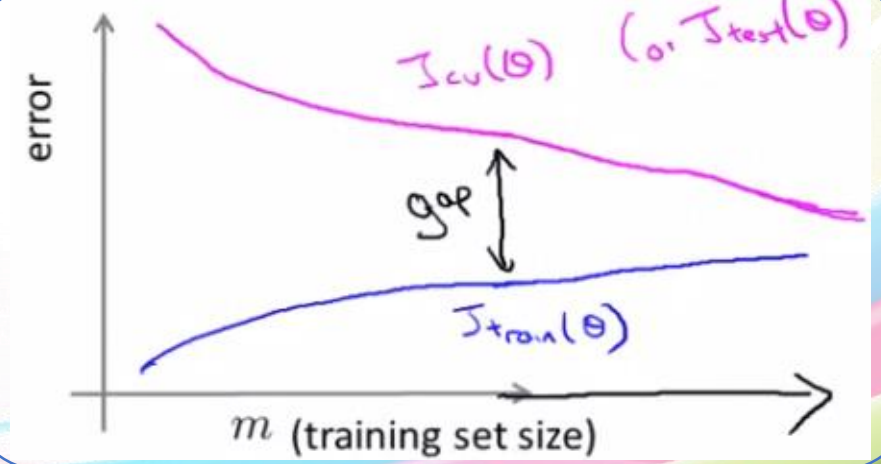
Bias vs. Variance

Learning Curve

- Thus, from the previous slide this can be conclude that:

> If a learning algorithm is suffering from high variance, getting more training data is likely to help.

- Thus, following figure depicts that increasing training data size can help as it can bring down cross-validation or test error.



- Thus, plotting training and validation errors is quite helpful to figure out if your learning algorithm is suffering from high bias or high variance.

Bias vs. Variance

Learning Curve

- In the following two circumstances, getting more training data is likely to significantly help a learning algorithm's performance:
  1. Algorithm is suffering from high variance
  2. $J_{CV}(\theta)$ (Cross-validation error) is much larger than $J_{train}(\theta)$ (training error)

Bias vs. Variance

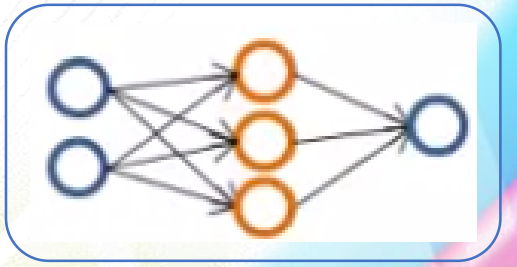
Deciding What to Do Next Revisited

- Suppose you have implemented regularized learning regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

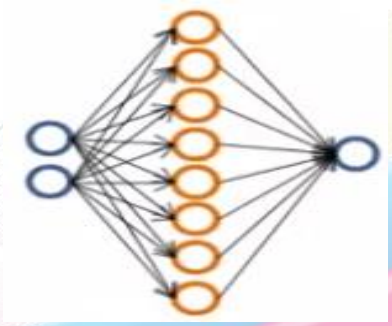
1	Get more training examples	Fixes high variance
2	Try smaller sets of features	Fixes high variance
3	Try getting additional features	Fixes high bias
4	Try adding polynomial features (x_1^2, x_2^2, x_1, x_2, etc.)	Fixes high bias
5	Try decreasing λ	Fixes high bias
6	Try increasing λ	Fixes high variance

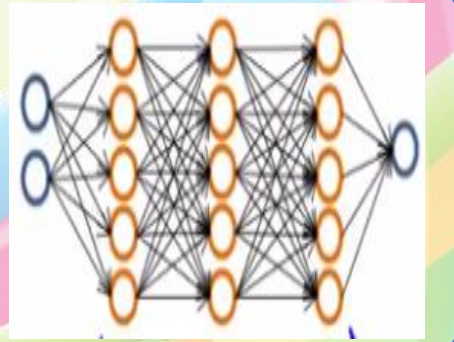
Bias vs. Variance

Deciding What to Do Next Revisited



- Small neural networks, i.e., fewer parameter, are prone to underfitting, but computationally cheaper.





- Large neural networks, i.e., more parameters, are prone to overfitting and computationally expensive. To address overfitting, use regularization parameter λ.

- Often, it is better to have large neural networks as you can solve the issue of overfitting by regularization. Besides this, using regularization to reduce / resolve overfitting is better than using smaller neural networks. However, the only potential drawback of this strategy is computation overhead.
- To figure out the appropriate number of hidden layers, utilise the technique of cross-validation.

Bias vs. Variance

Deciding What to Do Next Revisited

- Suppose you fit a neural network with one hidden layer to a training set. You find that the cross-validation error $J_{CV}(\theta)$ is much larger than the training error $J_{train}(\theta)$. Is increasing the number of hidden units is likely to help?
  - No, because it is currently suffering from high variance, so adding hidden units is unlikely to help.

Bias vs. Variance

Error Analysis

- If you are start building any machine learning system or model, following are the recommended steps:
  - Start with a simple algorithm that you can implement quickly, then test it on your cross-validation data.
  - Plot learning curves to decide if more data, more features, etc., are likely to help.
  - Error Analysis: Manually examine the examples (in cross-validation set) that your algorithm made errors on. See if you can spot any systematic trend in what type of examples it is making errors on.
- Using gut feeling to tune / optimize our machine learning model is called as pre-mature optimization. We should make use of learning curve, error analysis, etc., to make decision. “Let evidence guide your decisions rather than gut feeling”.
- Skewed Classes: Skewed Classes are scenarios where the number of observations belonging to one class are significantly lower than those belonging to other class. These occur commonly in electricity pilferage, fraudulent bank transaction, etc.
- Following are some error metrics for skewed classes:

		Actual Classes	
		1	**0**
Predicted Classes	**1**	True Positive	False Positive
	0	False Negative	True Negative

Bias vs. Variance

Error Analysis

- In a Skewed Data set, usually we consider class is 1 for the class with less instances or rare occurrences in real life.
- Precision:

  Of all instances that we have predicted as positive (i.e., class 1), how many actually are positive.

$$\text{Precision} = \frac{True\ Positives}{Number\ of\ predicted\ positives} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- Recall:

  Of all instances that are actually positive (i.e., class 1), how many have we predicted as positive.

$$\text{Recall} = \frac{True\ Positives}{Number\ of\ Actual\ Positives} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Bias vs. Variance

Trading off Precision and Recall

- Suppose, we have a Logistic Regression classifier for cancer disease where:

$$0 \leq h(x) \leq 1$$

A usual way of prediction is:

Predict 1 (i.e., cancer) if $h(x) \geq 0.5$

Predict 0 (i.e., no cancer) if $h(x) < 0.5$

However, predicting cancer for someone is a very sensitive thing as that person will go through an extremely expensive and painful diagnosis and treatment. Thus, we want to be very sure before giving prediction of cancer for any patient. Therefore,

Predict 1 (i.e., cancer) if $h(x) \geq 0.7$

Predict 0 (i.e., no cancer) if $h(x) < 0.7$

In the above case, we will get high precision, but lower recall.

Suppose, we have further strengthen the threshold values to make our prediction for cancer more accurate and confident, then:

Predict 1 (i.e., cancer) if $h(x) \geq 0.9$

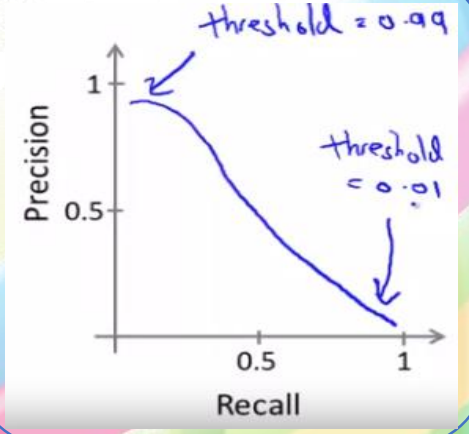
Predict 0 (i.e., no cancer) if $h(x) < 0.9$

This will further increase the precision and decrease the recall.

Bias vs. Variance

Trading off Precision and Recall

- Now, due to high threshold value, there may be a case where we predict no cancer, i.e., 0, for a patient actually having cancer, i.e., 1. Thus, in this case that patient will get relaxed and never go for further treatment and diagnosis, and die after some time. Therefore, we also want to reduce false negative.



To address this issue, we will do something similar:

Predict 1 (i.e., cancer) if $h(x) \geq 0.3$

Predict 0 (i.e., no cancer) if $h(x) < 0.3$

In the above case, we will get higher recall and lower precision.

Thus, this is called as the trade off between precision and recall.

Based on the requirements of your problem, you should select the threshold value:

Predict 1 (i.e., cancer) if $h(x) \geq$ threshold_value

Predict 0 (i.e., no cancer) if $h(x) <$ threshold_value

This gives rise to a question: is there a way to automatically select this threshold_value?

Or how do we compare different precision and recalls?

Now, suppose we have three algorithms and their precision and recalls. How to decide which one of them is best?

	Precision (P)	Recall (R)
Algorithm 1	0.5	0.4
Algorithm 2	0.7	0.1
Algorithm 3	0.02	1.0

Bias vs. Variance

Trading off Precision and Recall

- To figure out the best algorithm out of the given algorithms in the last slide, we need a single number to compare their performance as comparing two values (precision and recall is not possible.
- To resolve this, we can take average of precision and recall, and then compare the average of all three algorithms. However, these averages will be biased towards higher precision or recall, hence will not help us to find out the best optimized algorithm which will be balanced for both precision and recall.
- Thus, there is a different way of combining precision and recall, called as F1 Score or simply F score:

$$\text{F1 Score (or simply F score)} = 2 \text{ x } \frac{(Precision\ x\ Recall)}{(Precision + Recall)}$$

- Following is the comparison of previously mentioned algorithms:

	Precision (P)	Recall (R)	~~Average~~	F1 Score (or simply F Score)
Algorithm 1	0.5	0.4	~~0.45~~	0.444
Algorithm 2	0.7	0.1	~~0.4~~	0.175
Algorithm 3	0.02	1.0	~~0.51~~	0.0392

A perfect F score will be 1 when precision and recall will be 1. Whereas, the worst F score will be 0. Thus,

$0 \leq \text{F Score} \leq 1$

Bias vs. Variance

Data for Machine Learning

- It is not recommended to blindly go and spend a lot of time collecting a huge amount of data as only sometimes having a huge amount of data can be helpful.
- Only under certain kind of condition, having a huge amount of data and training a learning algorithm over it can be quite effective.
- Consider the following two problems:
    1. We want to train an algorithm that can select the right word from a given set of confusing words. For instance, {two, to, too}, {then, than}, etc. Thus, the right word for the following sentence is “two”: Today, I ate ___ eggs.
    2. Predicting house price provided only one feature is given: size of house in square feet.

    So, in the first problem, an English language expert with a huge knowledge (or data) can perform well. Whereas, in the second problem, even an extremely experienced broker will not able to tell house prices precisely based on their sizes as other features, like number of bedroom, location, etc., also matters a lot.
- Thus, having a lot of data is not necessary to get good performance. Sometimes, features are also important.

Bias vs. Variance

Data for Machine Learning

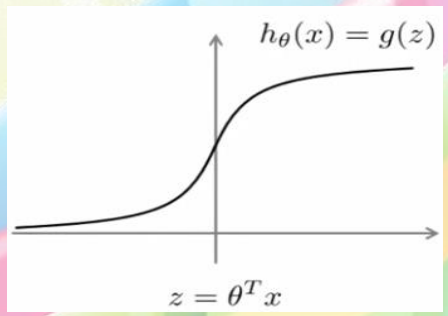
- Following are the recommendations:
  1. If you are using a learning algorithm with many parameters (for instance, logistic regression / linear regression with many features; neural network with many hidden units, i.e., low bias algorithms), then use very large training set as these are low bias algorithm. Thus, their $J_{train}(\theta) \approx J_{CV}(\theta) \rightarrow$ small $J_{test}(\theta)$, i.e., low variance. Consequently, the system will become of low bias and low variance learning algorithm (desired and ideal).

Large Margin Classification

Optimization Objective

- Support Vector Machine (SVM) is used in both, industry and academia.
- Sometimes, SVM gives better and cleaner result than both, Linear / Logistic Regression / Random Forest and Neural Network.
- We will start learning this algorithm (i.e., SVM) with its optimization objective.
- To understand SVM's optimization objective, we will start with Logistic Regression and modify it a bit to get Support Vector Machine (SVM).
- In Logistic Regression, the hypothesis function looks like:

$$h(x) = \frac{1}{1+ e^{-\theta^T x}}$$



If y = 1, we want h(x) ≈ 1, $\theta^T x >> 0$ (i.e., $\theta^T x$ need to be much greater than 0)

If y = 0, we want h(x) ≈ 0, $\theta^T x << 0$ (i.e., $\theta^T x$ need to be much smaller than 0)

Cost of Logistic Regression can be given by:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log h(x^i) + (1 - y^i) \log(1 - h(x^i))\right]$$

So, here the contribution in loss by one training instance can be given by:

$$\underbrace{-y \log\left(\frac{1}{1+ e^{-\theta^T x}}\right)}_{\text{Term 1}} \underbrace{- (1 - y) \log\left(1 - \frac{1}{1+ e^{-\theta^T x}}\right)}_{\text{Term 2}}$$

Large Margin Classification

Optimization Objective

- Now, when y = 1, the second part of the last equation (i.e., Term 2) will become zero due to "(1 – y)".
- Similarly, when y = 0, the first part of the last equation (i.e., Term 1) will become zero due to "y".
- In case of y = 1, following is the plot of Term 1:
  - In this figure, $z = \theta^T x$.
  - z is on x-axis and Term 1 is on y-axis.
  - When, z is large, the value of Term 1 is very small but greater than 0.
  - Thus, for any $(x^{(i)}, y^{(i)})$, if the value of $y^{(i)} = 1$, Logistic Regression tries to set z (i.e., $\theta^T x$) large.

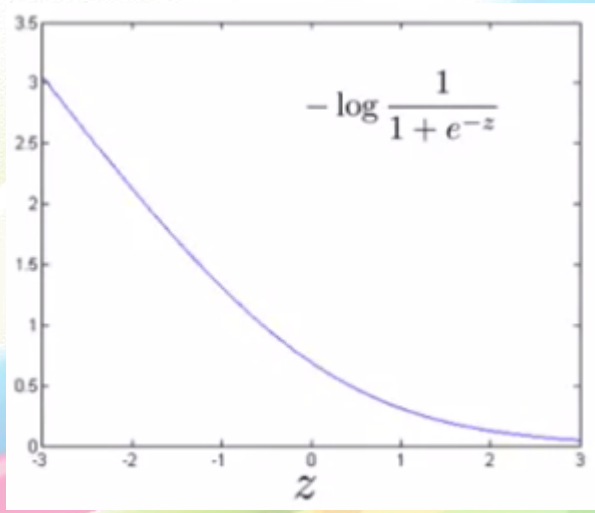


Fig. Plot for Term 1 in case of y = 1

- In case of y = 0, plot for Term 2 is given in left.

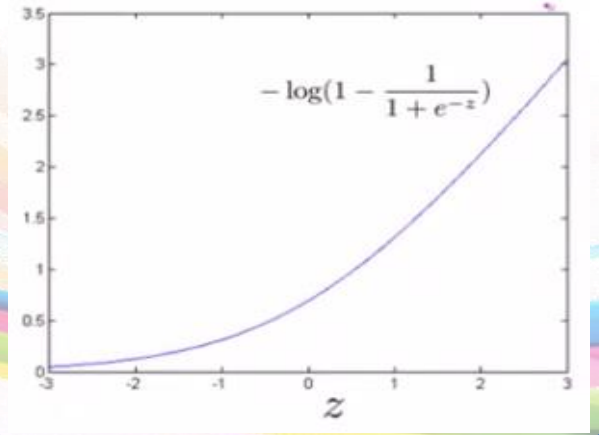


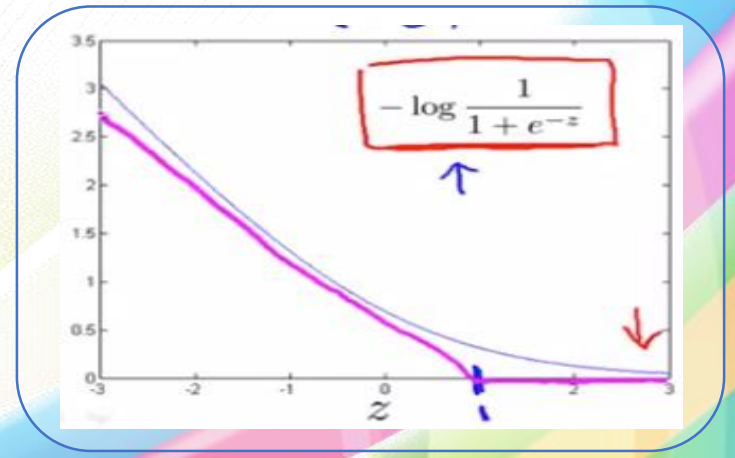
Fig. Plot for Term 2 in case of y = 0

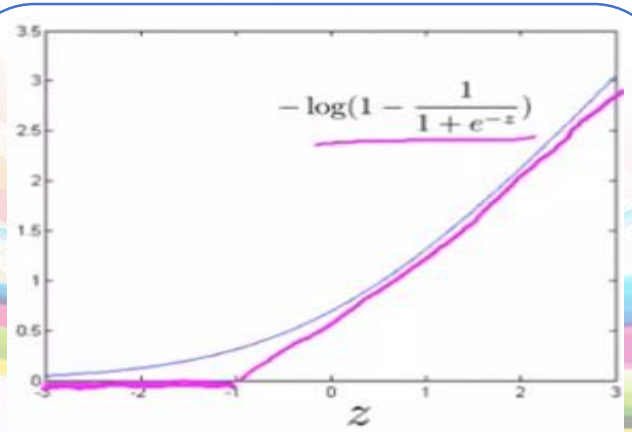
Large Margin Classification

Optimization Objective

- In case of y = 1, the term that remain in cost function of Logistic Regression is: $-log\left(\frac{1}{1+e^{-z}}\right)$, where z = $\theta^T x$

- The above term can be modified to have cost function of SVM. The plot of modified cost function will look like this (shown in purple colour):
- In the given plot, there are two line segments, one at the right, i.e., the flat portion, and the another at the left inclined at some angle.
- Let us call the function of this plot (in purple colour) as $cost_1(z)$, 1 in subscript denotes that this is the plot when y = 1.





- Similarly, when y = 0, we will get second term from the cost function of Logistic Regression. We can modify it to get cost function of SVM. Following is its plot (in purple colour):
- Same as above, this plot has two line segments, one is flat and another is inclined at same angle.
- Let us call the function of this plot (in purple colour) as $cost_0(z)$, 0 in subscript because y = 0 for this part.

Large Margin Classification

Optimization Objective

- Following is the cost function of Logistic Regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} \underbrace{y^i \log h(x^i)}_{\text{Term 1}} + \underbrace{(1 - y^i) \log(1 - h(x^i))}_{\text{Term 2}}\right] + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2}_{\text{Regularization Term}}$$

- For SVM, we will replace the Term-1 of above equation by $cost_1(z)$ and Term-2 by $cost_0(z)$, where $z = \theta^T x$.
- Following is the general form of cost function of SVM:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \, cost_1(\theta^T x^{(i)}) + (1 - y^i) \, cost_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

- By convention, we write cost function of SVM slightly differently. We will ignore the constant (1/m) because the problem of finding minimum doesn't get affected by the multiplication of a constant.
- Generally, the format of cost function is like this: A + λB, where A is the actual cost function (i.e., Term-1 and Term-2 as shown in above equation), λ is the regularisation parameter and B is some other term. However, the format of cost function for SVM is: CA + B, where C is the regularisation parameter, A is the actual cost function (i.e., Term-1 and Term-2) and B is some other term.
- In case of Logistic Regression, setting a high value for λ implies high values of B. Whereas, in case of SVM, small value of C implies large values of B.
- Thus, you can consider C as equivalent of $\frac{1}{\lambda}$

Large Margin Classification

Optimization Objective

- Following is the cost function of SVM (notice, that we have rid of (1/m) term mentioned in the cost function of Logistic Regression and replaced (1/λ) by C):

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{i} cost_1\left(\theta^T x^{(i)}\right) + (1 - y^{(i)} cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

- Unlike Logistic Regression, SVM doesn't output probability; instead, following is the hypothesis of SVM:

$$h_\theta(x) = 1 \; (if \; \theta^T x \quad 0); otherwise, 0$$

Large Margin Classification

Large Margin Intuition

- Here, we will see that why SVM is called as Large Margin Classifier.
- As we know that following is the cost function of SVM:

$$\min_{\theta} C \sum_{i=1}^{m} [y^i cost_1(\theta^T x^{(i)}) + (1 - y^{(i)} cost_0(\theta^T x^{(i)})] + \frac{1}{2}\sum_{i=1}^{n} \theta_j^2$$

- Figure 1 is the plot of $cost_1(z)$ and figure 2 is the plot of $cost_0(z)$.
- In these figures:
  - If y = 1, we want $\theta^T x \geq 1$ (not just ≥ 0)
  - If y = 0, we want $\theta^T x \leq -1$ (not just < 0)
- Now, as $\theta^T x \geq 1$ for y = 1 and $\theta^T x \leq -1$ for y = 0, thus there is a huge gap between 1 and -1. This is a large margin due to which SVM is called as Large Margin Classifier.

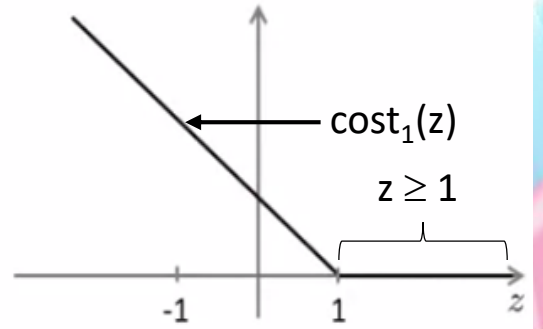


Figure 1

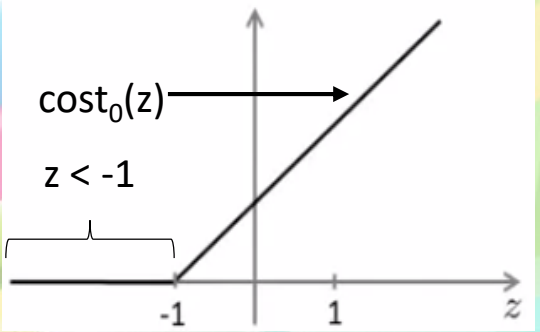
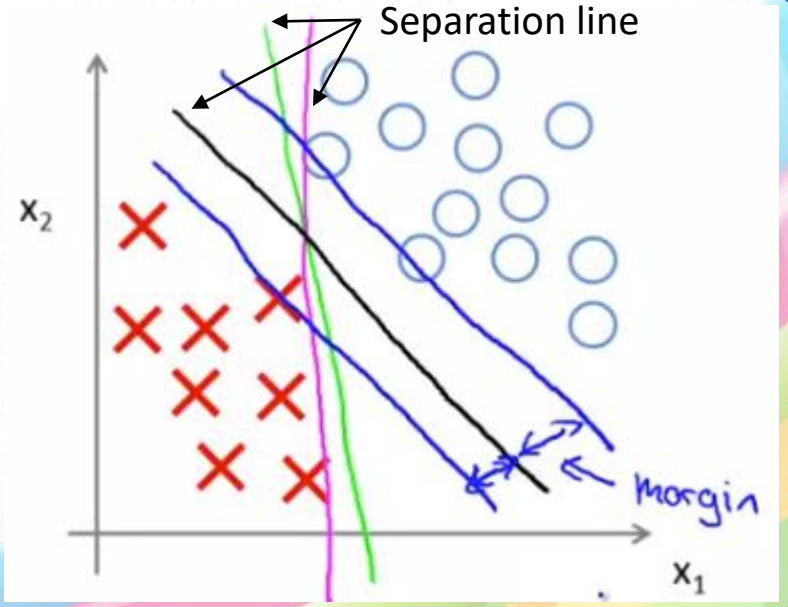


Figure 2

Large Margin Classification

Large Margin Intuition

- The given figure is visually depicting the concept of large margin classifier.
- In this figure, bubbles and red crosses are representing data points of positive and negative classes.
- These data points can be separated by using any of below separation line:
  1. Purple separation line
  2. Green separation line
  3. Black separation line
- However, if see carefully, we may find that the black separation line is the best separation line as it maintains maximum distance from the closest point of both the classes (i.e., positive and negative).

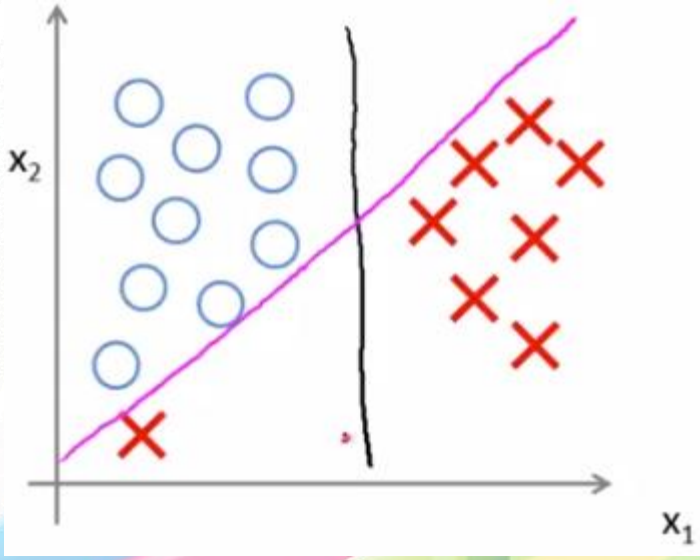


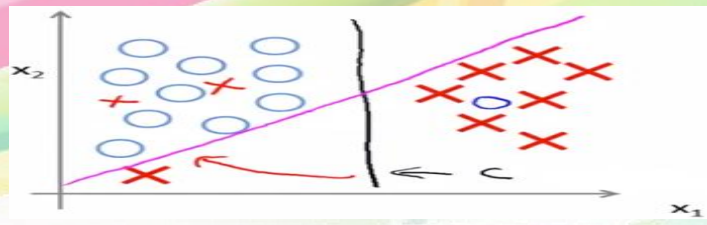
- SVM always find the best separation line, i.e., black color separation line in this case.
- The distance of closest points of both the classes from the best separation line is called as margin.
- Thus, SVM is called as Large Margin Classifier.

Large Margin Classification

Large Margin Intuition

- The given figure is showing the case of outliers.
- Suppose, there is an outlier which is present towards another class, red-cross near bubbles, in this case; then, will that be a good idea to change the decision boundary from black to purple only for few outliers (one in this case).
- Since we don't have idea of this thing, thus, we can control it by adjusting the value of regularization parameter C:
  - If C is very large, it will change the decision boundary from black to purple.
  - If C is not too large, then we will end up with black decision boundary.
- However, if there are many outliers, as shown in the below figure, then SVM does the right thing by resolving it.





Large Margin Classification

Mathematics Behind Large Margin Classification

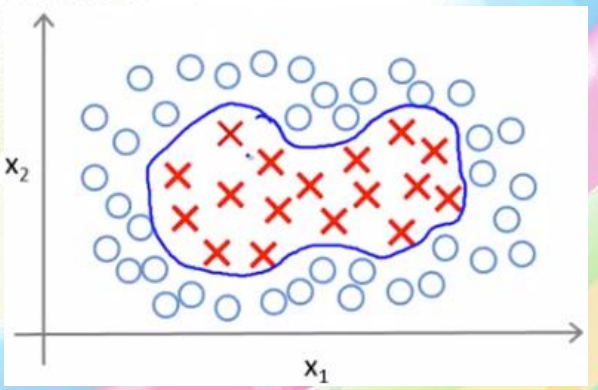
Skipped

- Vector Inner Product
- SVM Decision Boundary

Kernels

Kernels – I

- Here, we will see how to use SVM as non-linear classifier.
- The main technique for doing that is called as Kernels. Let us see what are Kernels and how to use them.
- If the data points of our data look something like as shown in the figure, then we need a non-linear classifier to separate them.
- One way to do so is to come up with a set of complex features, something like this:



Predict y = 1 if,

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \ldots. \geq 0$$

Predict y = 0 if,

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \ldots. < 0$$

- Thus, we can also write complex features in the form of functions, something like this:

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \theta_5 f_5 + \ldots.$$

Where, f_i is denoting a function of features.

- Now, question arise: is there a different / better choice of features?

Since we are going to deal with a lots of features, especially in case of images, thus in that case such high order and complex combination of features may not help us, as well as shoot the computation requirements.

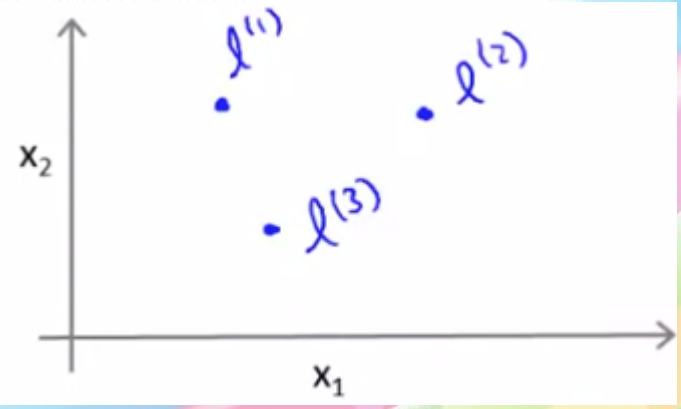
Kernels

Kernels – I

- Here is one idea for how to define new features. For this, we will see an example in which we will define three new features, however for real problems, there can be much more features.
- In the given figure, we have chosen three data points, let us call them landmarks ($l^{(1)}$, $l^{(2)}$, $l^{(3)}$), out of all data points.
- For any given data point x, we will calculate the similarity of x from these landmarks as shown below:

$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$f_3 = \text{similarity}(x, l^{(3)})$$

  The above mentioned "similarity" function can be Euclidean distance as well.



- This "similarity" function is called as Kernel.
- There are various types of Kernels, like Gaussian kernel, etc.
- Sometimes, instead of writing "similarity", we write "k" to denote kernel, like this:

$$f_1 = k(x, l^{(1)})$$
$$f_2 = k(x, l^{(2)})$$
$$f_3 = k(x, l^{(3)})$$

Kernels

Kernels – I

- Gaussian Kernel is one of many kernels. For any given point x, the gaussian kernel will define the similarity between x and landmark $l^{(1)}$ as:

$$\text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

In the above expression, the term in the numerator is nothing but the formula of Euclidean distance, and the overall expression is Gaussian Kernel.
If the given data point x is similar/closed to landmark $l^{(1)}$, then the distance would be less and exp(- small no.) ≈ 1.
Whereas, if the given data point x is not similar/closed to landmark $l^{(1)}$, then the distance would be large and exp(- large no.) ≈ 0.

- Now, we will see the impact of σ^2 on the above expression:

Figure 1

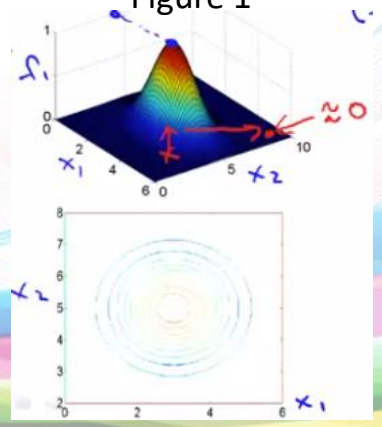


Figure 2

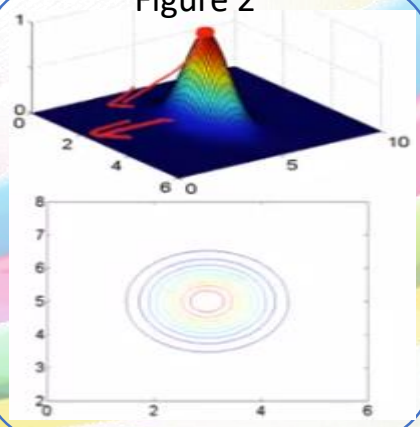
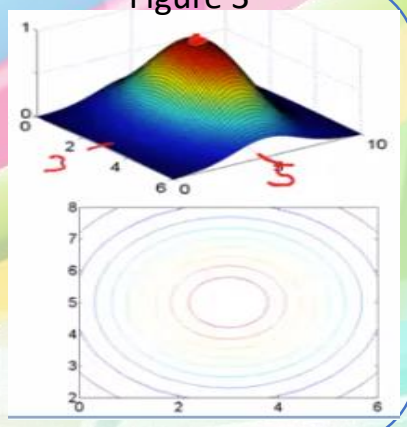


Figure 3



Kernels

Kernels – I

- Now, the value of Gaussian Kernel for every x_i and every $l^{(j)}$ can be considered as another feature f_j:

$$f_j = \exp\left(-\frac{||x_i - l^{(j)}||^2}{2\,\sigma^2}\right)$$

- Now, for the figure 1 of previous slide:
  - $\sigma^2 = 1$
  - Two attributes x_1 and x_2 are on horizontal axis.
  - The vertical axis is f_1.
  - A data point can be plot as per his (x_1, x_2) value and distance from landmark $l^{(1)}$.
  - A data point at the bottom will have $f_1 = 0$, means close to landmark $l^{(1)}$.
  - Figure at bottom is the contour of above 3D graph.

- Now, for the figure 2 of previous slide:
  - $\sigma^2 = 0.5$, steeper than fig1.
  - Two attributes x_1 and x_2 are on horizontal axis.
  - The vertical axis is f_1.
  - A data point can be plot as per his (x_1, x_2) value and distance from landmark $l^{(1)}$.
  - A data point at the bottom will have $f_1 = 0$, means close to landmark $l^{(1)}$.
  - Figure at bottom is the contour of above 3D graph.

- Now, for the figure 3 of previous slide:
  - $\sigma^2 = 3$, broader than fig1.
  - Two attributes x_1 and x_2 are on horizontal axis.
  - The vertical axis is f_1.
  - A data point can be plot as per his (x_1, x_2) value and distance from landmark $l^{(1)}$.
  - A data point at the bottom will have $f_1 = 0$, means close to landmark $l^{(1)}$.
  - Figure at bottom is the contour of above 3D graph.

Kernels

Kernel – I

- In the given figure, we have three landmarks $l^{(1)}$, $l^{(2)}$ and $l^{(3)}$.
- Following is our hypothesis function:

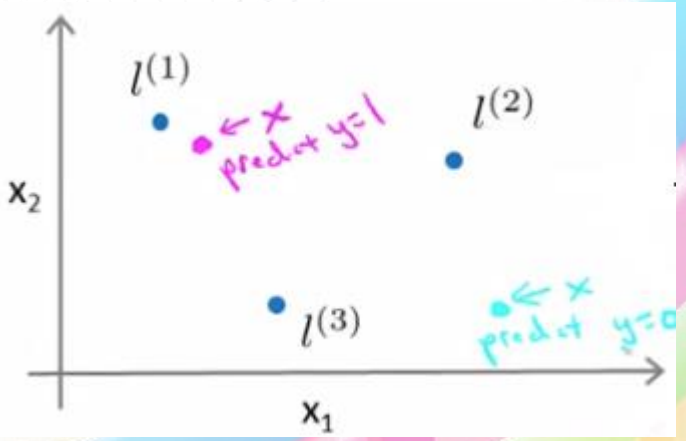
  Predict 1 when:

  $$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

  Otherwise 0.

  Here, $\theta_0 = -0.5$, $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = 0$, and

  $$f_j = \exp\left(-\frac{||x_i - l^{(j)}||^2}{2\sigma^2}\right)$$



- Suppose we have a point x (in Purple) for which we want to make prediction.
- Since x (in Purple) is closed to $l^{(1)}$ and far from $l^{(2)}$ & $l^{(3)}$, hence $f_1 \approx 1$, $f_2 \approx 0$, $f_3 \approx 0$. This gives us:

  $-0.5 + 1 = 0.5 \geq 0 \Rightarrow$ Prediction 1
- Similarly, another point x (in Cyan colour), is far from all these landmarks ($l^{(1)}$, $l^{(2)}$, $l^{(3)}$), thus $f_1 \approx 0$, $f_2 \approx 0$ and $f_3 \approx 0$. This implies:

  $-0.5 + 0 = -0.5 < 0 \Rightarrow$ Prediction 0

Kernels

Kernels – II

- Here, we will see that how we get landmarks.
- In a complex learning problem, we may have much more landmarks than simply three.
- Following is the process of choosing landmarks:

Given: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots\ldots, (x^{(m)}, y^{(m)})$

Choose: $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, l^{(3)} = x^{(3)}, \ldots., l^{(m)} = x^{(m)}$

For any given x (x can be from training / validation / test data), we will calculate similarity by using kernel function:

f_1 = similarity(x, $l^{(1)}$)
f_2 = similarity(x, $l^{(2)}$)
:
:
f_m = similarity(x, $l^{(m)}$)

} Creating a feature vector, $f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$, here $f_0 = 1$

Thus, for any $x^{(i)}$, having features $\{x^{(i)}_1, x^{(i)}_2, \ldots, x^{(i)}_m\}$, we will get $f^{(i)}$ $\{f^{(i)}_1, f^{(i)}_2, \ldots, f^{(i)}_m\}$, where:

$f^{(i)}_1$ = similarity($x^{(i)}$, $l^{(1)}$)
$f^{(i)}_2$ = similarity($x^{(i)}$, $l^{(2)}$)
:
$f^{(i)}_m$ = similarity($x^{(i)}$, $l^{(m)}$)

In case, $(x^{(i)}, y^{(i)})$ is a training instance, in that case one of $f^{(i)}$ will be one due to $x^{(i)} = l^{(i)}$.

Kernels

Kernels – II

- Thus, we have transformed $x^{(i)}$ ($x^{(i)}_0$, $x^{(i)}_1$, $x^{(i)}_2$, …., $x^{(i)}_m$) into $f^{(i)}$ ($f^{(i)}_0$, $f^{(i)}_1$, $f^{(i)}_2$, …., $f^{(i)}_m$), where $x^{(i)}_0 = 1$ and $f^{(i)}_0 = 1$.
- So, $f^{(i)}$ is our new feature representation of actual $x^{(i)}$.

$$x^{(i)} \rightarrow f^{(i)}$$

- Following is the procedure of SVM with Kernels:

  Hypothesis: Given x, compute features f.

  Predict y = 1, if $\theta^T f \geq 0$, where $\theta^T f = \theta_0 f_0 + \theta_1 f_1 + \theta_2 f_2 + \ldots\ldots + \theta_m f_m$

  During training, SVM minimizes the below cost function:

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^i cost_1\left(\theta^T f^{(i)}\right) + (1 - y^{(i)} cost_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

  Earlier, $x^{(i)}$ was provided in the cost function of SVM, now that $x^{(i)}$ is replaced by $f^{(i)}$ in the above expression.

  Additionally, n = m because of our new features $f^{(i)}$ in place of $x^{(i)}$.

  The last term (i.e., $\frac{1}{2}\sum_{i=1}^{n} \theta_j^2$) is a bit different than this to improve computations; which need not to know at this time.

- One would be thinking to use the kernels of SVM for other algorithms, like Logistic Regression, however that is not possible as the computation tricks that apply for SVM cannot be generalize for other algorithms.

Kernels

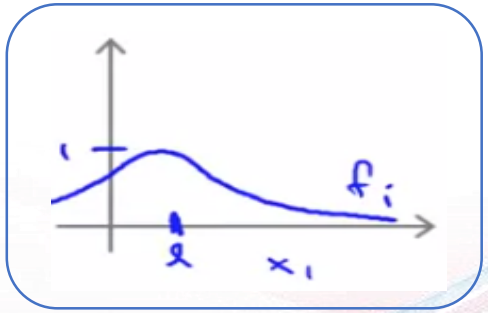
Kernels – II

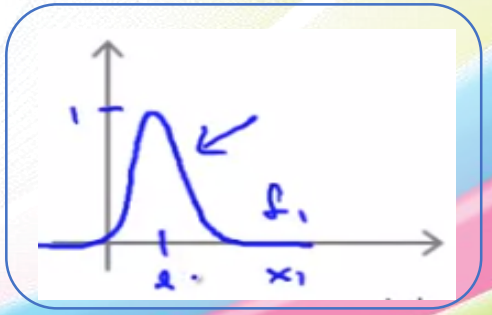
- SVM parameters:

$$C\left(=\frac{1}{\lambda}\right): \begin{cases} Large\ C: Lower\ Bias, High\ Variance \\ Small\ C: Higher\ Bias, Low\ Variance \end{cases}$$

$$\sigma^2: \begin{cases} Large\ \sigma^2: Features\ fi\ vary\ more\ smoothly; Higher\ Bias, Lower\ Variance \\ Small\ \sigma^2: Features\ fi\ vary\ less\ smoothly; Lower\ Bias, Higher\ Variance \end{cases}$$

More smooth curve





Less smooth curve

- Suppose you train an SVM and find it overfits your training data. Which of these would be a reasonable step?

  Decrease C and Increase σ^2.

Kernels

Using An SVM

- Instead of writing any optimization code by yourself, it is recommended to use software libraries written by researchers after hard work of years with unmatched mathematical optimization. Some of such libraries are: liblinear, libsvm, etc., to solve for parameter θ.
- Despite having optimization libraries, one need to choose:
  - Parameter C
  - Choice of Kernel (similarity function): If you do not choose any Kernel, there will be by default "linear kernel". Linear Kernel is like a linear classifier.
  - In case of Gaussian Kernel, one has to choose σ^2 and also perform feature scaling. Feature scaling is important because values of some features can be very large, in that case SVM will give more weightage to those features and less / no weightage to features with small values. For instance, value of "area" (viz 1000, 1200, 900, etc.) is much larger than "number of bedrooms" (viz 1, 2, 3, 4, 5).
  - Not all similarity functions make valid kernels. A similarity kernel need to satisfy a technical condition called Mercer's Theorem to make sure that SVM packages run correctly and do not diverge.
  - Many off-the-shelf kernels: Polynomial Kernel, String Kernel, Chi-Square Kernel, etc.
- Suppose you are trying to decide among a few different choices of kernel and also choosing parameters, such as C, σ^2, etc. How would you make the choice?

  Choose whatever performs better on the cross-validation data.

Kernels

Using An SVM

- Multi-class Classification
  - Many SVM packages already have in-built multi-class classification functionality.
  - Otherwise, use one-vs-all method.
- Logistic Regression vs SVM:
  - If n (number of features) is much larger than m (number of training examples), then use Logistic Regression or SVM without kernel (Linear Kernel).
  - If n is small and m is intermediate, use SVM with Gaussian Kernel.
  - If n is small, but m is large, then create / add more features, then use Logistic Regression or SVM without a kernel.
  - Neural Network is likely to work well for all the above conditions, but may be slower to train.

Clustering

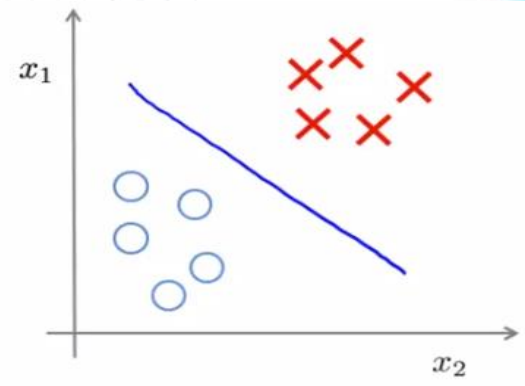
Unsupervised Learning: Introduction

Supervised Learning

- In Supervised Learning (under classification), we have provided with a data having labels with them.

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots., (x^{(m)}, y^{(m)})\}$$

- The job of Supervised Learning for classification problems is to find a decision boundary that separate the data of different classes.
- So, the supervised learning problem is: given a set of labels and fitting a hypothesis to it.

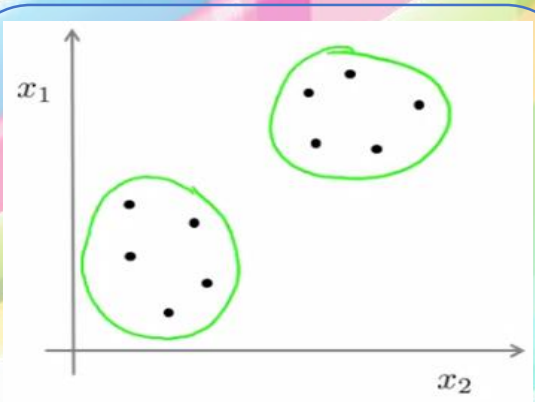


Unsupervised Learning >> Clustering Algorithm

- In Unsupervised learning, we do not have any labels with data.

$$\{x^{(1)}, x^{(2)}, \ldots., x^{(m)}\}$$

- Here, we pass this kind of unlabeled data set to unsupervised learning algorithm and ask it to provide us some structure in the data set.
- Suppose algorithm has provided us two groups in the data set, as shown in the figure, then this is called as Clustering Algorithm under Unsupervised Learning.



Clustering

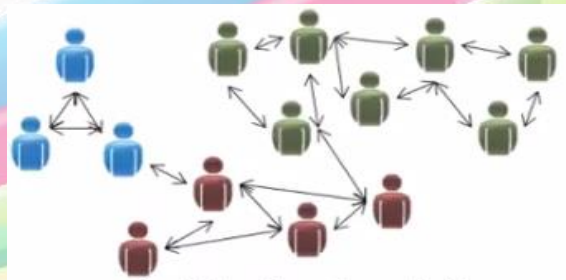
Unsupervised Learning: Introduction

- Clustering Algorithm is the first type of Unsupervised Learning Algorithm that we will cover.
- There are several other types of Unsupervised Learning Algorithm as well.
- Following are some of the applications of Clustering:

Market segmentation

Organize computing clusters

Astronomical data analysis

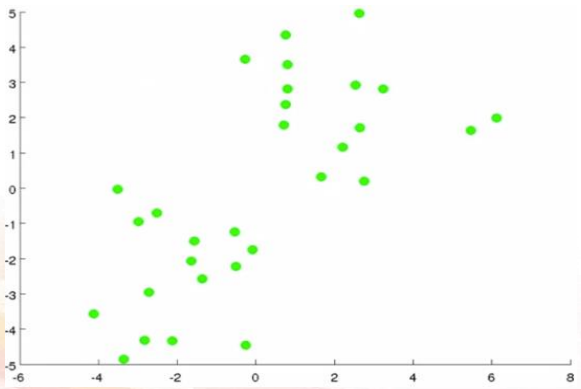


Social network analysis

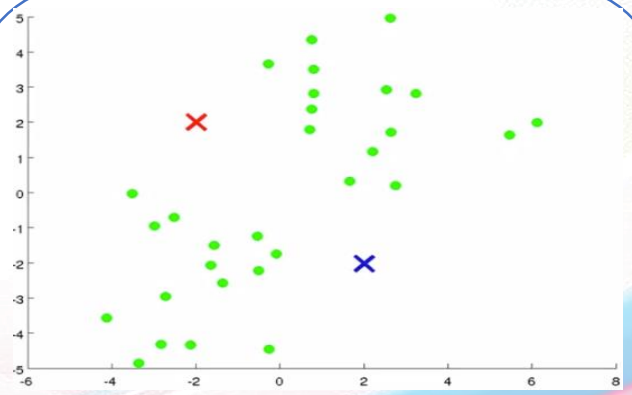
Clustering

K-Means Algorithm

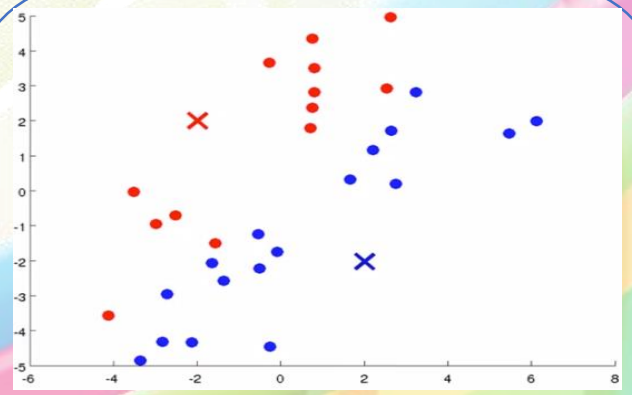
- K-Means is the most popular clustering algorithm.
- K-Means is an iterative algorithm and it does two things:
  1. Cluster assignment step
  2. Move centroid step



Suppose, initially this is how our data set looks.



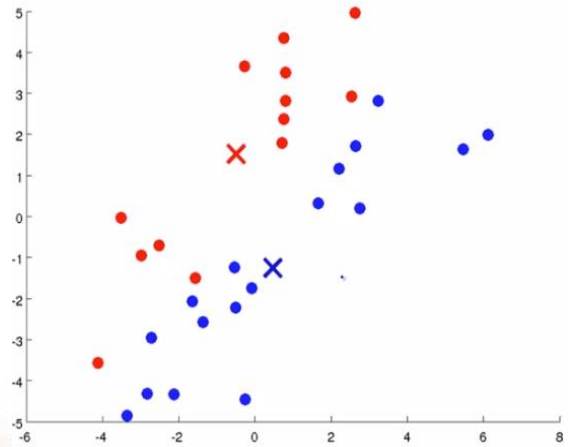
Clustering algorithm will randomly select two points as centroids, shown here in red and blue color crosses.



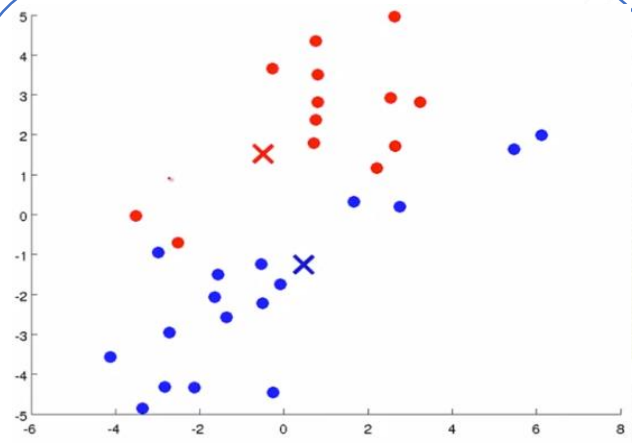
Then, the algorithm will calculate the distance of each data point from these centroids and based on that it will assign them their group, either red or blue, i.e., cluster assignment step.

Clustering

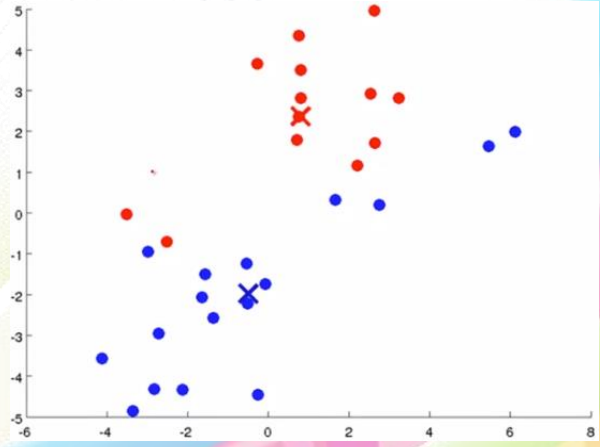
K-Means Algorithm



Now, in the second iteration, algorithm will mathematically calculate the centroids of two groups, “Move Centroid Step”.



Now, algorithm again calculates distance of each data point from new centroids and assign them new clusters, “Cluster Assignment Step”.



After this, algorithm again calculate new centroids as per new clusters and move the centroids, “Move Centroid Step”.

- The above iterative process, comprised of steps “Cluster Assignment Step” and “Move Centroid Step”, will continue until the centroids change. After a few iterations, centroid will not change and cluster will get fixed.

Clustering

K-Means Algorithm

- Following is the pseudocode of K-Means algorithm:

Input:

KMeans(number of clusters)

Training Set $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

Information:

Drop the convention $x_0 = 1$, *thus* $x^{(i)} \in R^n$, *there will not be the convention* $x_0 = 1$, *as earlier.*

Algorithm:

Randomly initialize K cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\} \in R^n$

Repeat {

Cluster Assignment Step

for I =1 to m:

$c^{(i)}$ *:= index (from 1 to K) of cluster centroid closest to* $x^{(i)}$

Move Centroid Step

for k = 1 to K:

μ_k *:= average(mean) of points assigned to cluster k*

}

Clustering

K-Means Algorithm

- So far, we have seen well separated clusters, like this (Figure 1):

Figure 1

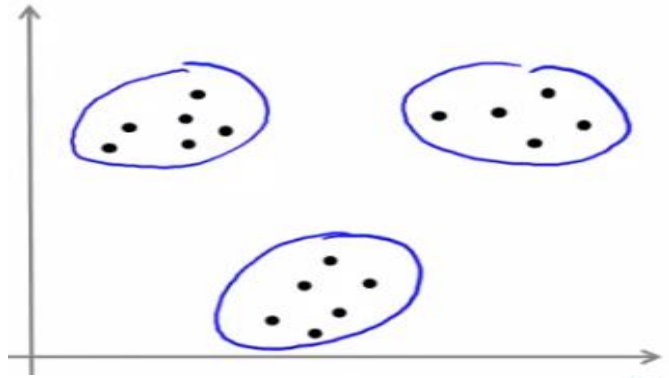
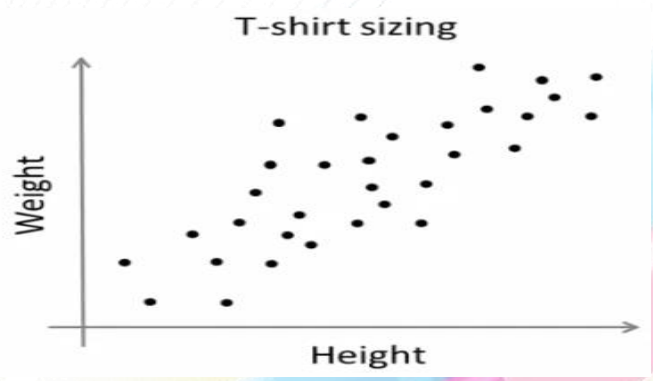


Figure 2



- Very often, Kmeans also apply on data which is not well separated, like shown in figure 2.
- "T-shirt sizing" figure is showing the plot of different sizes of t-shirts. If one wants to create groups of these sizes for Small (S), Medium (M) and Large (L), then Kmeans can do the job.
- Figure 3 is the output of Kmeans on T-shirt sizing data set shown in figure 2.
- Kmeans work in the similar manner for other market segmentation problems.

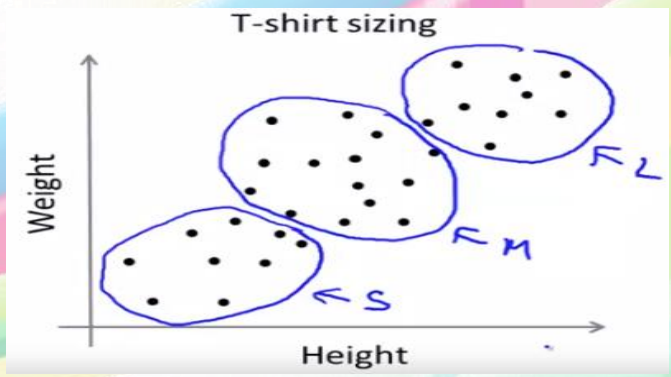


Figure 3

Clustering

Optimization Objective

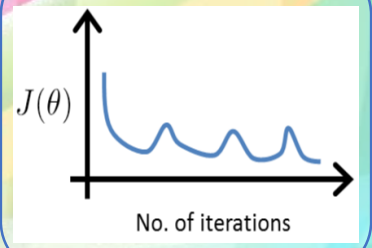
- Like Supervised Learning algorithms (like Logistic Regression), K-Means also has an optimization / cost function.
- Following are some notations:
  - $c^{(i)}$: Index of cluster (1, 2, …., K) to which example $x^{(i)}$ is currently assigned.
  - μ_k: k^{th} cluster centroid
  - K: Total number of clusters
  - $\mu_c(i)$: Cluster centroid of cluster to which example $x^{(i)}$ has been assigned
- Following is the optimization / cost function:

$$J\left(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k\right) = \frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - \mu_c(i)||^2$$

$$\min_{(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_k)} J\left(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k\right)$$

This cost function is also called as Distortion Function.

- Following is how K-Means algorithm works:
  1. The first step, "Cluster Assignment Step" (explained earlier), minimizes J wrt ($c^{(1)}$, $c^{(2)}$, …, $c^{(m)}$) keeping (μ_1, μ_2, …, μ_k) fixed.
  2. The second step, "Move Centroid Step" (explained earlier), minimizes J wrt (μ_1, μ_2, …, μ_k).
- It is not possible for the cost function to sometimes increase as shown in the figure.



Clustering

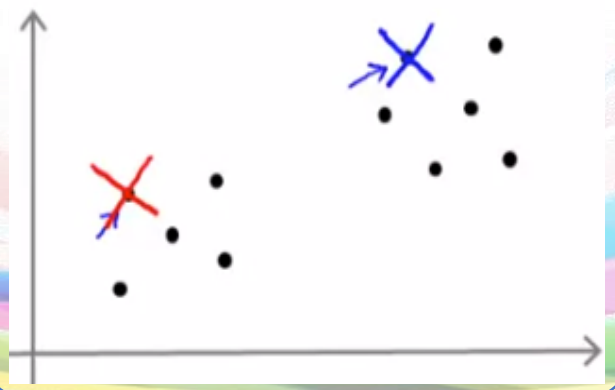
Random Initialization

- Here, we will discuss:
  1. How to initialise K-Means.
  2. How to make K-Means avoid local optima as well.
- There are several methods to randomly initialize K cluster centroids, however there is one method that is much more recommended than others.
- First of all, there is a restriction that the number of kernels should be less than the number of data points:

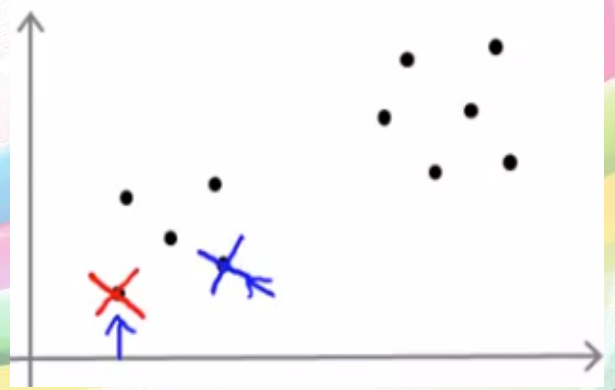
$$K < m$$

- Randomly picking any K data points as the cluster centroids can be good, as well as bad; as shown in the below figure:

Good Initialization



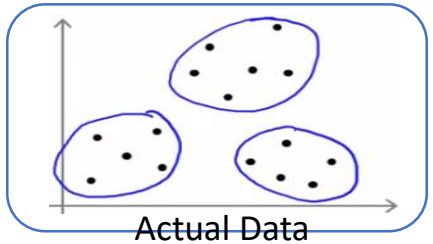
Bad Initialization

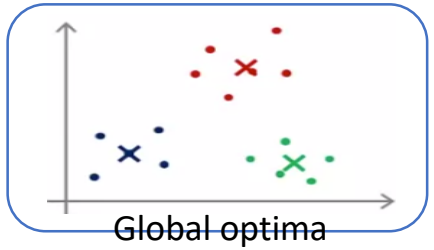


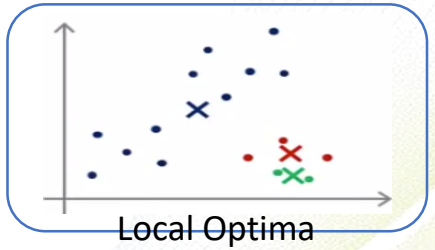
Clustering

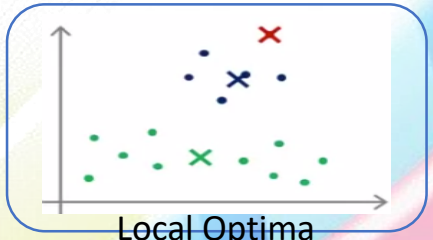
Random Initialization

- Thus, based on initialization, K-Means may end different solutions and can take different number of iterations.
- For instances:


Actual Data


Global optima


Local Optima


Local Optima

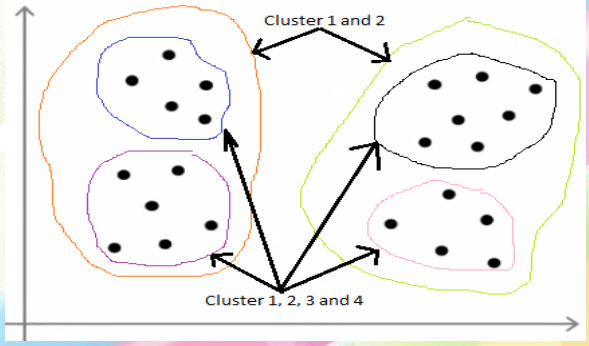
  - In above figures, cross marks are showing cluster centroids.
  - First figure is showing the actual clusters in data.
  - Second figure is showing the optimal solution with centroids at the global optimum location.
  - Third and fourth figures are showing the local optimum values of centroids.
- If total number of clusters (i.e., K) is small (around 2 to 10), then running the K-Means algorithm multiple times with randomly initialised centroids may result in the best solution.
- However, if the number of clusters (i.e., K) is large, then running the K-Means algorithm multiple times is less likely to give the best or optimum solution.
- Thus, the recommended way of initializing cluster centroids is:

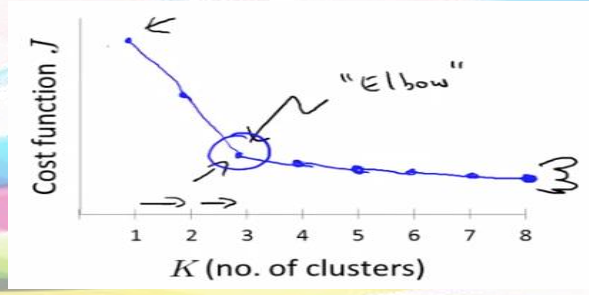
  Pick k distinct random integers $i_1, i_2, \ldots, i_k$ from {1, 2, …., m} and set $\mu_1 = x^{(i1)}$, $\mu_2 = x^{(i2)}$, …., $\mu_k = x^{(ik)}$.

Clustering

Choosing the Number of Clusters

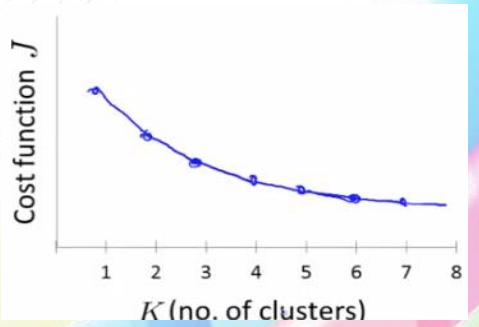
- The most general and popular method of choosing the number of clusters is by looking at the visualisation of data, i.e., manually.
- Looking at the visualisation of data and finding the number of clusters can be ambiguous. For instance, some may see two clusters and some may see four clusters in the given figure:
- Thus, finding number of clusters can be quite confusing.
- One method to find the number of clusters in a given data set is Elbow Method.
- Elbow Algorithm:
  - Creates a plot of cost (or distortion in case of K-Means) function against number of clusters.
  - The point where algorithm finds elbow of plot is considered as the K.





Clustering

Choosing the Number of Clusters

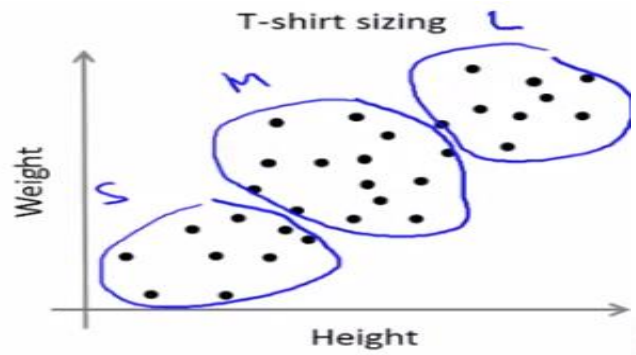


- Sometimes, it is hard to find elbow in a plot. For instance, same is shown in this plot.
- Elbow can be at 3 or 4 or might be at 5.
- Thus, in most of the situations, the location of elbow is not clear, and hence the optimal number of clusters is hard to find.
- Therefore, one should try Elbow method to figure out the right number of clusters, however on should not have much expectations from this method to find the right number of clusters.
- Suppose you run k-means using k=3 and k=5. You find that the cost function J is much higher for k=5 than for k=3. What can you conclude?
  - ☐ This is mathematically impossible. There must be a bug in the code.
  - ☐ The correct number of clusters is k=3.
  - ☑ In the run with k=5, k-means got stuck in a bad local minimum. You should try re-running k-means with multiple random initialization.
  - ☐ In the run with k=3, k-means got lucky. You should try re-running k-means with k=3 with different random initializations until it performs no better than with k=5.

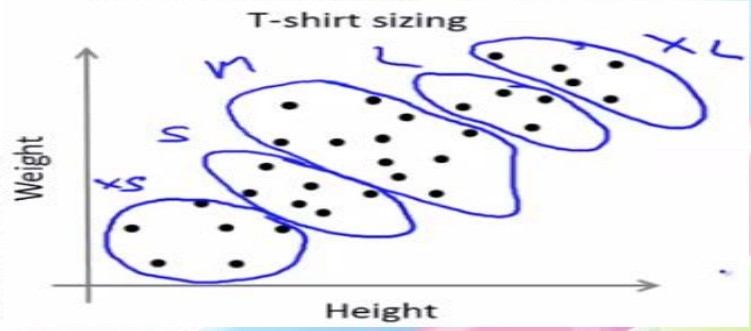
Clustering

Choosing the Number of Clusters

- Sometimes, you are running k-means to get clusters to use for some later / downstream purpose. Evaluate k-means based on a metric for how well it performs on that later purpose.
- For instance:



- Here, we have considered 3 clusters: S, M and L.
- A person can choose 3 clusters based on his capability to make t-shrits of different sizes.
- Producing t-shirst of less number of sizes may reduce overhead of having different machines on the manufacturer.



- Here, we have considered 5 clusters: XS, S, M, L & XL.
- A person can choose 5 clusters based on his desires to offer maximum fitting to his customers.
- This will make his customers happy and may raise his income.

Clustering

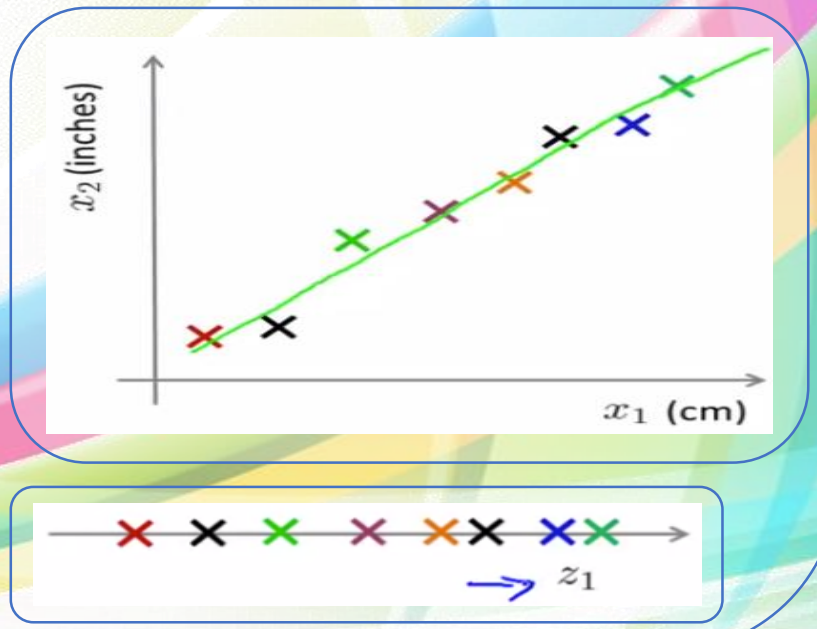
Choosing the Number of Clusters

- To summarize, the number of clusters k is mostly chosen by hand based on some later purpose or insights.
- However, one systematic approach can be Elbow method, but it does not work well every time.
- The best way to choose number of clusters is to ask yourself the purpose of k-means and decide the number of clusters.

Dimensionality Reduction

Motivation I: Data Compression

- One of the biggest advantage of Dimensionality Reduction is Data Compression.
- Data Compression provides us following benefits:
  - Less space on hard disk
  - Less bandwidth while uploading and downloading
  - Less processing for our data processing algorithms
- Given plot is showing data in two dimensional, x_1 and x_2.
- Here, x_1 and x_2 are the two features of data.
- Suppose, we want to reduce this two-dimensional data into one-dimensional.
- For this, we have figured out a straight line which is capturing almost every point in their plot, i.e., x_1 along horizontal and x_2 along vertical.
- Let us call this straight line which is capturing all data points, as z_1.
- The reduced dimension, z_1, will have corresponding points of all actual data points. This can be done by dropping a perpendicular on z_1 line.
- This z_1 is a straight line, as shown in the below figure.
- Thus, this is how we have reduced our two-dimensional data into one-dimensional.



Dimensionality Reduction

Motivation I: Data Compression

- Practically, we may have a thousand or more dimensionality data and we want to reduce it to a few 100s of dimensions.
- Here, we will see one more example of reducing data from three-dimensionality to two-dimensionality.

Figure 1

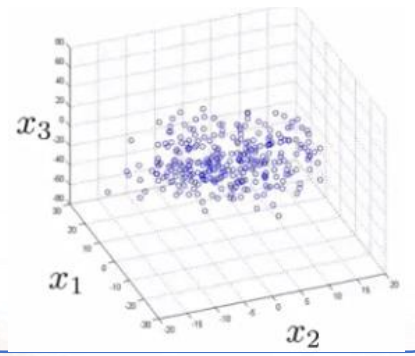


Figure 2

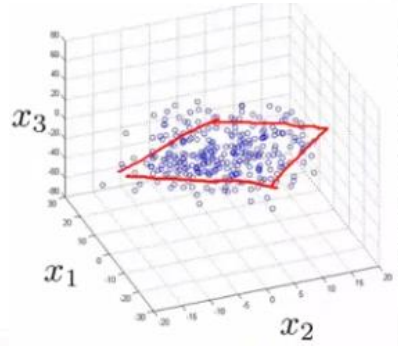
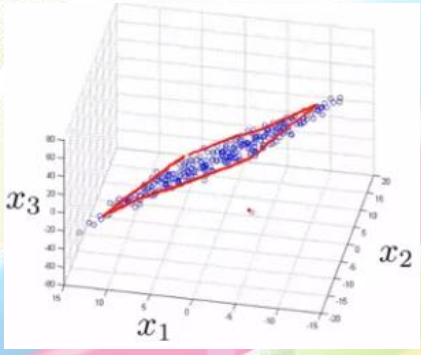


Figure 3



- In figure 1, a three dimensional data is shown with three features (or dimensions), namely x_1, x_2 and x_3.
- In figure 2, it is shown that the entire three-dimensional data is approximately projected on a two-dimensional plane.
- Due to this, now we need only two features (or dimensions), namely z_1 and z_2, to locate any data point, as shown in figure 3.

Dimensionality Reduction

Motivation I: Data Compression

- Suppose, we apply dimensionality reduction to a dataset of m examples $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, where $x^{(i)} \in R^n$. As a result of this, we will get out:
  - ❑ A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\}$ of k examples where $k \leq n$.
  - ❑ A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\}$ of k examples where $k > n$.
  - ✓ ❑ A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in R^k$ for some value of k and $k \leq n$.
  - ❑ A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in R^k$ for some value of k and $k > n$.

Dimensionality Reduction

Motivation II: Visualization

- The second application of Dimensionality Reduction is Data Visualization.
- For a lot of Machine Learning algorithms, it is required to understand data first before applying the algorithm as it helps us to make quick improvements.

Figure 1

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

Figure 2

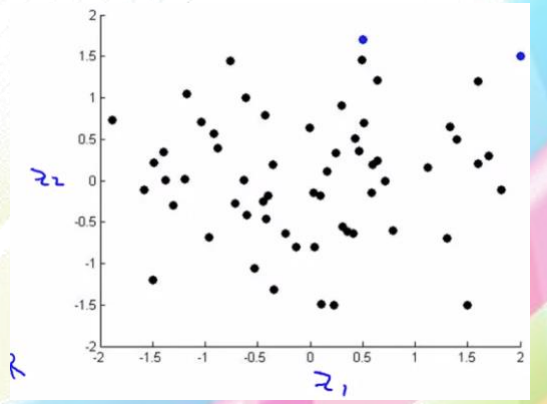
Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

- In the figure 1, we have a lot of features, like GDP, Per Capita GDP, Human Development Index, etc. If somebody wants to visualise this data, then it would not be possible as the data has more than three dimensions.
- In figure 2, it is shown that by applying some method we have reduced multiple features into only two features, z_1 and z_2, thus now it will be easy for us to visualize the data.

Dimensionality Reduction

Motivation II: Visualization

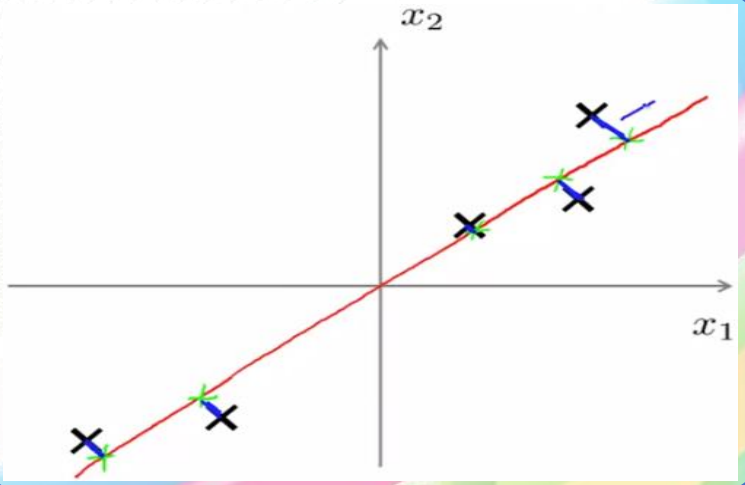
- This figure is showing the two-dimensional plot of countries for reduced dimensions (or features), i.e., z_1 and z_2.
- In this plot, each dot is representing one country (or instance of previous data set).



Principal Component Analysis (PCA)

Principal Component Analysis Problem Formulation

- Principal Component Analysis (PCA) is the most popular algorithm for Dimensionality Reduction.
- Following is the formulation of a problem which can be solved by PCA (refer the given figure):
  - In this figure, we have black cross marks which are representing our actual data points having two features (or dimensions).
  - We want to reduce dimensionality and bring it down to one dimension, i.e., on a straight line, shown in red color.
  - We find that green marks on red line are the projections of our actual data point represented in red color crosses.
  - Distance between actual data points and projected data points is quite less or minimum.
  - Thus, what PCA does?
  - PCA tries to find a surface (a line in this case) where the distance between the actual data points and the projected data points remain very less or minimum.
  - This distance (shown in little blue line) can be measured in sum of squares. This distance is sometimes also called as the Projection Error.

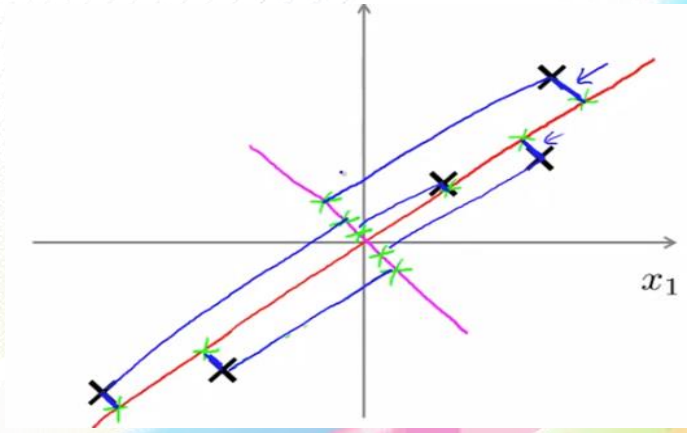


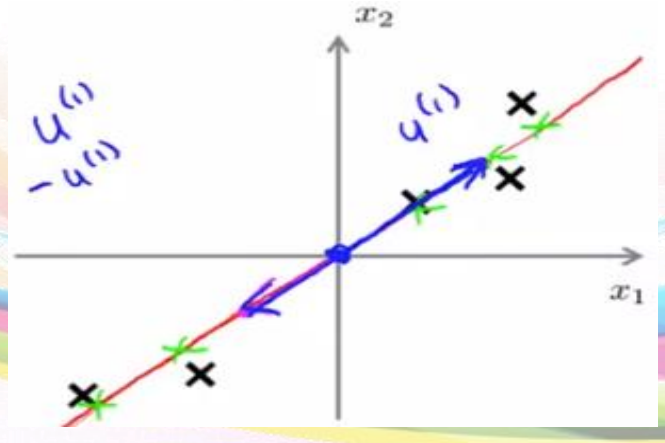
- Before applying PCA, it is a standard practice to perform mean normalization and feature scaling.

Principal Component Analysis (PCA)

Principal Component Analysis Problem Formulation

- Similarly, just assume if we choose purple line as our projection surface, then it is clear that the distance of our actual data points from the projected points on the purple line would be very huge.



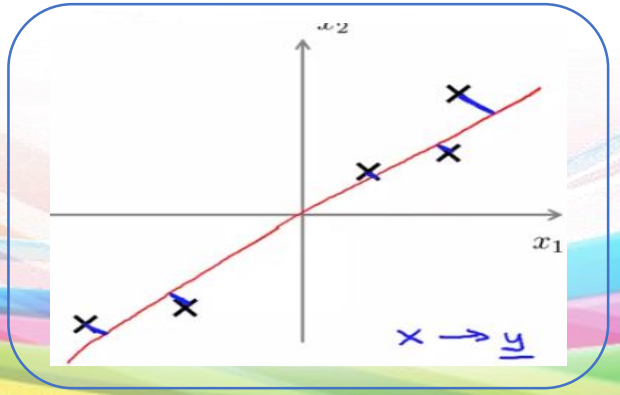


- PCA will provide us a vector, let us call it $\mathbf{u^{(1)}}$.
- We have to programmatically extend that vector in both the directions to find the line on which actual data points will projected.
- It doesn't matter if the vector $\mathbf{u^{(1)}}$ positive or negative (i.e., in the reverse direction) as in both the case, we will extend it further in both the direction.

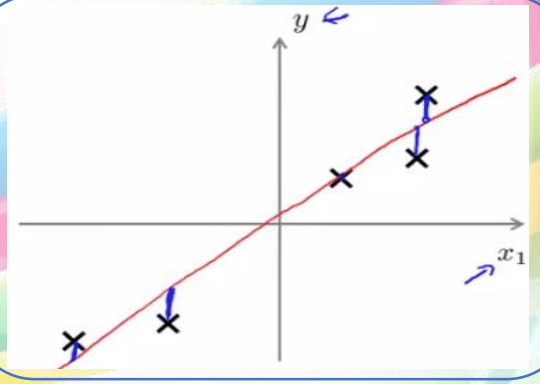
Principal Component Analysis (PCA)

Principal Component Analysis Problem Formulation

- Generally, we have to reduce dimensionality of data from n-dimensions to k-dimensions to project the data so as to minimize the projection error.
- For this, we will find k vectors: $\mathbf{u}^{(1)}$, $\mathbf{u}^{(2)}$, …., $\mathbf{u}^{(k)}$.
- PCA and Linear Regression have some similarity, but it doesn't mean that PCA is Linear Regression or vice-versa.
- In Linear Regression, we find a line and calculate its vertical distance (as shown by some blue vertical offset) from the line. This is shown in the given figure (at bottom-right).
- Whereas, in case of PCA, this distance is calculated by using perpendicular offsets, as shown in the below bottom-left figure.



PCA – perpendicular offset



Linear Regression – vertical offset

Principal Component Analysis (PCA)

Principal Component Analysis Algorithm

- Before applying PCA, always do pre-processing of data as it is a standard procedure.
- Following is the description:

  Training Set: $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}$

  Pre-processing of data: Feature Scaling (optional – depends on the nature of data) / Mean Normalization

  $$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

  Replace each $x^{(i)}_j$ with $x_j = \mu_j$

  } Mean Normalization

  If different features are on different scales (for instance, x_1 is size of houses, x_2 is number of bedrooms, etc.), then scale features to have comparable range of values, i.e., Feature Scaling.
- Reduce data from n-dimensions to k-dimensions:

  Compute Covariance Matrix (Σ):

  $$\Sigma = \frac{1}{m}\sum_{i=1}^{n} x^{(i)} x^{(i)^T}$$

  Note: Do not get confused by summation symbol. At LHS, it is representing Covariance Matrix (Σ) and at RHS it is representing summation.

  Computer Eigen Vectors of Covariance Matrix (Σ):

  [U, S, V] = svd(Sigma); # "svd"is a function to calculate Singular Value Decomposition in Matlab

  "svd" is Singular Value Decomposition.

Principal Component Analysis (PCA)

Principal Component Analysis Algorithm

- The U matrix looks as shown in the given figure.
- This matrix is a nxn matrix.
- To get the reduced dimensions, we need k vectors.
- These k vectors can be obtained by taking the first k columns of U matrix, denoted by $U_{reduced}$, a nxk matrix.

$$U = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix}}_{\text{k-columns as k-vectors}} \in \mathbb{R}^{n \times n}$$

- To reduce n-dimensions to k-dimensions, i.e.,

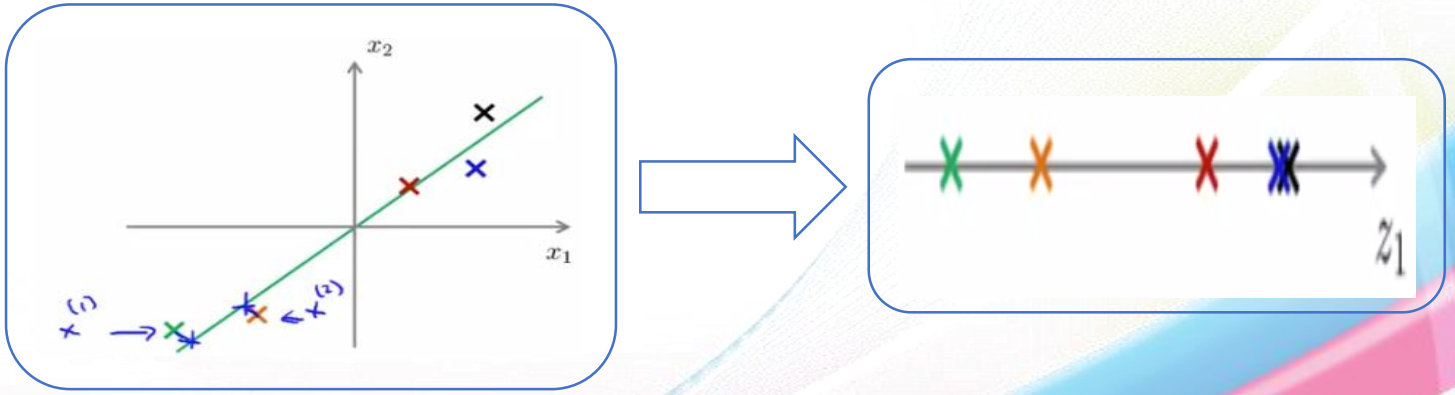
  $x \in R^n \rightarrow z \in R^k$ # reduce n-dimensional data x to k-dimensional data z

  $z = \underbrace{U_{reduced}^T}_{\text{A k-dimensional matrix}} \underbrace{x}_{\text{nx1 vector}}$

Applying PCA

Reconstruction from Compressed Representation

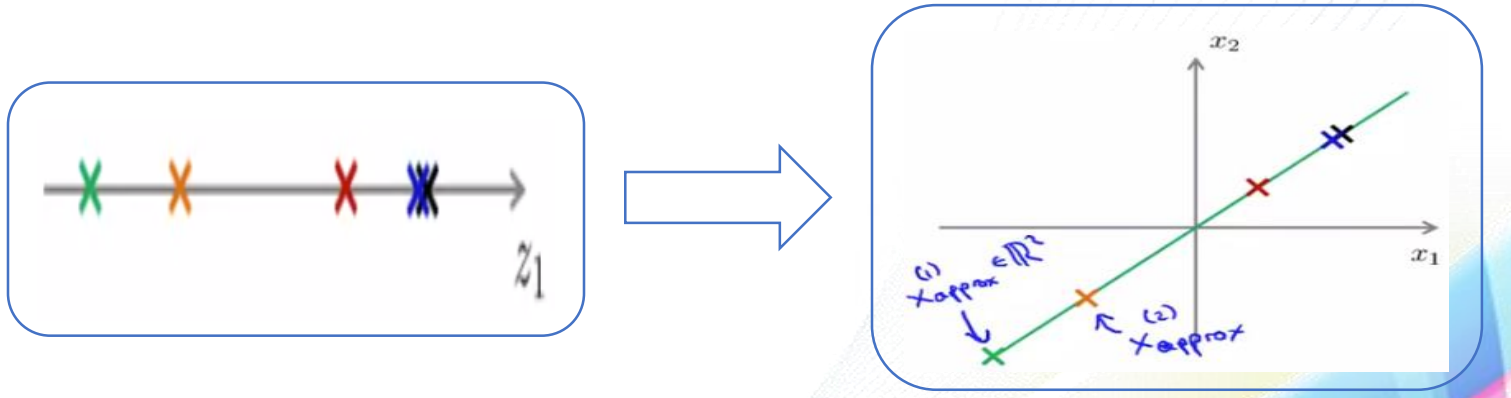
- Earlier, we have seen that PCA is a kind of compression algorithm.
- Then, there must be some way to get back the approximation of our original data from the compressed data that we obtained from PCA.



- Figure in left is demonstrating the transformation of our original data to a reduced dimensionality, i.e., on one dimensional line and figure at right side is showing that one dimensional line. Thus, following is the mathematical representation of this: $z = U_{reduce}^{T} x$
- Now, we want to find x from a given z.
- Following is the mathematical formula to achieve the same: $x_{approx} = U_{reduce}\, z$, where U_{reduce} is a nxk matrix and z is a kx1 vector, thus the product will give us a nx1 vector.

Applying PCA

Reconstruction from Compressed Representation



- In the above figures, it is depicted the transformation of PCA compressed data (i.e., reduced dimensionality data) to the approximate original data. All the data points on the line in figure at right are the approximations of original data points. This is pretty closed to the original data points.
- Suppose we run PCA with k = n, so that the dimension of the data is not reducecd at all (this is not useful in practice but is a good thought exercise). In this case, following will remain true:
  1. U_{reduce} will be a nxn matrix
  2. $x_{approx} = x$ for every example x
  3. The percentage of variance retain will be 100%

Applying PCA

Choosing the Number of Principal Components

- In PCA, we reduce our n-dimensional data to k-dimensional data (k<n). Thus, PCA takes k as an argument.
- The number k is also called as the number of principal component.
- Following is a useful concept to choose the number k:

  Average Squared Projection Error = $\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)} - x_{approx}^{(i)}\right|\right|^2$

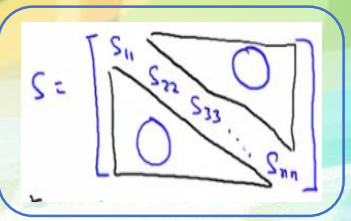
  Total variation in the data = $\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)}\right|\right|^2$

  Choose k as much minimum as possible, but satisfy the following condition:

  $$\frac{Average\ Squared\ Projection\ Error}{Total\ Variation\ in\ the\ Data} \leq 0.01 \qquad (1\%)$$

  Another way of saying the above condition is “to retain 99% of variance”.
- Other common value is 0.05 (or 5% or retaining 95% of variance). Sometimes, even we can go as low as 85% of variance. However, mostly it is preferable to retain 95 to 99% of variance.
- As we know that svd (Singular Value Decomposition) function of gives three values in output: [U, S, V]
- S is a diagonal matrix of nxn dimensions as shown in the figure:
- Thus, the above expression can be alternately and easily calculate from matrix S:

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01 \qquad \text{OR} \qquad \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$



Applying PCA

Advice for Applying PCA

- Sometimes, PCA can be use to speed-up a supervised learning algorithm.
- Suppose, we have a supervised learning problem:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots., (x^{(m)}, y^{(m)})$$

Assume that $x^{(i)}$ is a 10,000 dimensional feature. Due to this high dimensional feature vector, the learning algorithm will get drastically slow.

Fortunately, PCA can reduce the dimensionality of this kind of data and make our learning algorithms faster.

$$(x^{(1)}, x^{(2)}, x^{(3)}, \ldots., x^{(m)}) \in R^{10,000}$$

$$\downarrow$$

PCA

$$\downarrow$$

$$(z^{(1)}, z^{(2)}, z^{(3)}, \ldots., z^{(m)}) \in R^{1,000}$$

Thus, our new training set will look like this: $(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), (z^{(3)}, y^{(3)}), \ldots., (z^{(m)}, y^{(m)})$

- Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by PCA only on the training set. This mapping can be applied as well to the example $x^{(i)}_{cv}$ and $x^{(i)}_{test}$ in the cross-validation and test sets.
- Application of PCA:
  - Compression
    - Reduce memory needed
    - Speed-up learning algorithm

    } Choose k by % of variance retain
  - Visualization: k would be either 2 or 3 as things can be visualized either in two or three dimensions.

Applying PCA

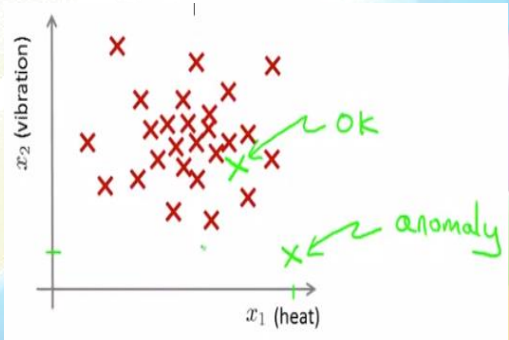
Advice for Applying PCA

- Bad use of PCA:
  - To prevent overfitting: Since $z^{(i)}$ has k features, whereas $x^{(i)}$ has n features ($k < n$). Thus, use $z^{(i)}$ instead of $x^{(i)}$. Due to less number of features in $z^{(i)}$, there will be less chances of overfitting.
  - Use of PCA to prevent overfitting might work okay, but it is not a good way to address overfitting. Use regularization instead.
- It is advisable to first try with the original data and then use PCA if it doesn't work.

Density Estimation

Problem Motivation

- Let us try to understand Anomaly Detection by an example:
  Suppose following are the features under consideration for the quality assurance (QA) testing for aircrafts in an assembly line:
  x_1: Heat Generated, x_2: Vibration Intensity, …..
  Following is our dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
  If we plot data points on a graph of heat vs vibration, then it may look something similar to this plot.
  For a new data point, x_{test}, it will be considered as accepted if it is in the crowd of other data points, however it will be considered as anomaly if it will be far from the accepted data points, as shown in this figure.



  To figure out if x_{test} is an anomaly or not, we will train a model p(x). We will label x_{test} as anomaly if $p(x_{test}) < \varepsilon$. Whereas, we will consider x_{test} as okay or accepted if $p(x_{test}) \geq \varepsilon$.
- Following are some of the applications of Anomaly Detection:
  1. Fraud Detection
  2. Manufacturing, like quality assurance, sensor values, etc.
  3. Monitoring computers in a data center, like memory use, network traffic, CPU usage, etc
- If your anomal detection system is flagging too many x as anomalous whenever $p(x) < \varepsilon$ that are not actually anomalous, then you should try decreasing ε.

Density Estimation

Gaussian Distribution

- Gaussian Distribution is also called as Normal Distribution.
- Say $x \in R$ (i.e., set of real numbers). If x is a Gaussian distribution with mean μ and variance σ^2, then this can be mathematically denoted as:

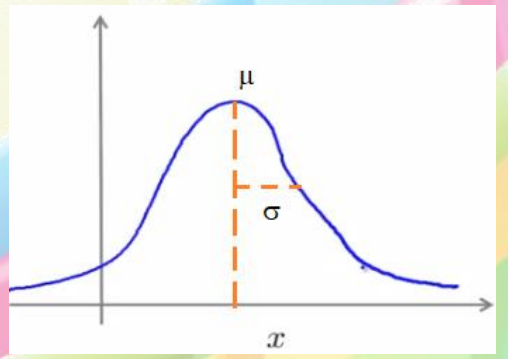
$$X \sim \aleph(\mu, \sigma^2)$$

Where, $\aleph$: Normal or Gaussian Distribution, ~: read it as "distributed as"

- Gaussian or Normal Distribution is a bell shaped curve.
- This curve or distribution is parametrized by mean μ and variance σ^2.
- The center of this bell curve is the mean μ and the width of this bell curve is standard deviation (σ).
- The value of x on both sides from the center keeps on decreasing. It shows either the probability of those values or the frequency of those values diminishes as we move away from the center in both the direction.
- Following is the formula of Gaussian or Normal Distribution:

$$P(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x- \quad)^2}{2\sigma^2}\right)$$

- The property of probability Gaussian Distribution is that the area under the curve must be 1.



Density Estimation

Anomaly Detection Algorithm

- Suppose, we have a dataset: $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

  Each data point from this dataset has n features: x $(x_1, x_2, x_3, \ldots, x_n)$

  Now, assume that each of these features are distributed as shown below:

  $x_1 \sim \aleph(\mu_1, \sigma_1^2)$

  $x_2 \sim \aleph(\mu_2, \sigma_2^2)$

  $x_3 \sim \aleph(\mu_3, \sigma_3^2)$

  :

  :

  $x_n \sim \aleph(\mu_n, \sigma_n^2)$

  Where, ~ denotes "distributed as".

  Thus, the probability of x can be given as:

  $p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2)\, p(x_3; \mu_3, \sigma_3^2) \ldots\ldots\ldots p(x_n; \mu_n, \sigma_n^2)$

  $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$

  $$p(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\,\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

  Anomaly if $p(x) < \varepsilon$

Building An Anomaly Detection System

Developing & Evaluating An Anomaly Detection System

- So far, we have considered anomaly detection problem as an unsupervised learning problem. However, now we will add one more feature, “label” (y = 0 if normal and y = 1 if anomaly).
- Assume that our training set has lots of training instances. We are considering all of them as non-anomalous or normal data point. It is okay to have a few anomalous data point in a huge training set.
- Let us say, we have 10,000 training data points with around 20 anomalous data points (unknowingly). In a data of 10,000 instances, it is okay to have from 2 to 50 anomalous data points that we will consider as non-anomalous during training.
- One can use cross-validation set to choose parameter ε.
- Following are the steps to evaluate an anomaly detection algorithm:
  1. Fit model p(x) on training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
  2. On a cross-validation / test set, predict:

$$y = \begin{cases} 1 & if\ p(x) < \varepsilon\ \ (anomaly) \\ 0 & if\ p(x) \geq \varepsilon\ \ (normal) \end{cases}$$

  3. Since data is very skewed (i.e., most of the lables are 0, “normal”), hence classification accuracy will not be a good choice to evaluate the performance of algorithm. Thus, it is advisable to use any of the below parameters to evaluate the performance of anomaly detection system:

1. True positive, False Positive, False Negative, False Positive
2. Precision / Recall
3. F_1 - Score

Building An Anomaly Detection System

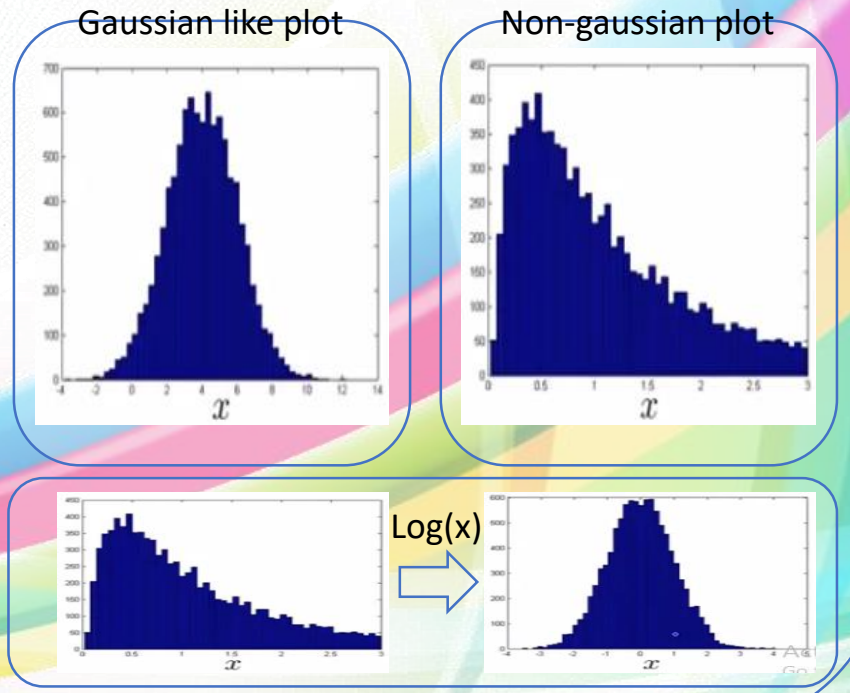
Anomaly Detection vs Supervised Learning

Anomaly Detection	Supervised Learning
1. Very small number of positive examples and very large number of negative examples.	1. Large number of positive and negative examples.
2. Many different types of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we have see so far.	2. Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.
3. Examples: Fraud Detection, Manufacturing, Monitoring Machines in a Data Center	3. Examples: Email Spam Classification, Weather Prediction (Summer, Rainy, etc.), Cancer Classification

Building An Anomaly Detection System

Choosing What Features To Use

- The performance of an anomaly detection system is greatly affected by the nature of features. Thus, it is crucial to see that what features one should use for their anomaly detection algorithm.
- As we have seen that in our anomaly detection system, we have modelled our features as Gaussian / Normal distribution: $p(x_i; \mu_i, \sigma_i^2)$.
- Thus, one of the things that one should do is to plot the histogram of data. The histogram plot of data should look like a normal / gaussian distribution. For instance, following plot ("gaussian like plot") can be considered as a normal distribution and feed into anomaly detection algorithm.
- However, if the histogram plot of data is like non-gaussian distribution as shown in the plot ("non-gaussian plot"), then one should apply some transformations to make it like gaussian or normal distribution.
- Anomaly detection algorithm works okay even if you don't transform your data into more gaussian, but it is better to transform. For instance, the following transformation operation can make "non-gaussian plot" into "gaussian plot".



Building An Anomaly Detection System

Choosing What Features To Use

- In anomaly detection, we want:
  - p(x) large for normal examples x.
  - p(x) small for anomalous examples x.
- A common problem in anomalous detection algorithm is that p(x) is comparably (say, both large) for normal and anomalous examples.
- To resolve the above issue, one needs to figure out instances which are incorrectly predicted by anomaly detection algorithm and try to create a new feature helpful in eliminating the issue.
- Suppose your anomaly detection algorithm is performing poorly and outputs a large value of p(x) for many normal and anomalous examples in your cross-validation dataset. In such case, it is wise to coming up with more features to distinguish between the normal and the anomalous examples.

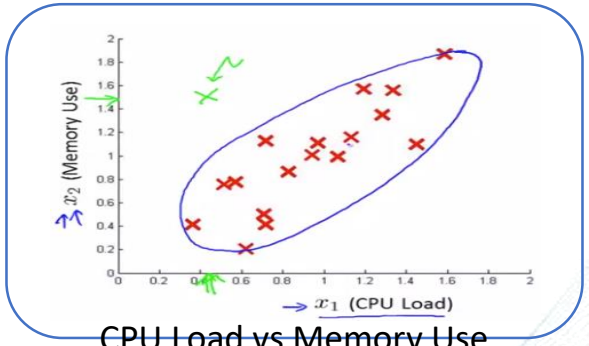
Multivariate Gaussian Distribution

Multivariate Gaussian Distribution

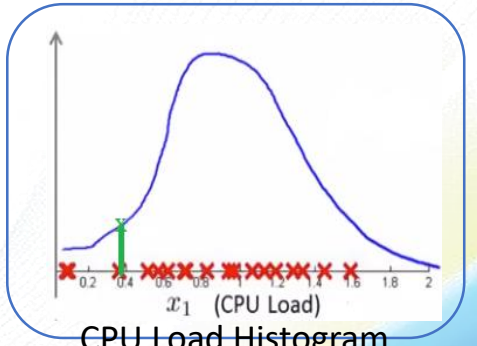
- Multivariate Gaussian Distribution works as an extension for Anomaly Detection Algorithm.
- Consider the following example to understand the need of Multivariate Gaussian Distribution:

There is a plot of CPU Load vs Memory Use. This plot has a green cross mark that looks like an anomaly.

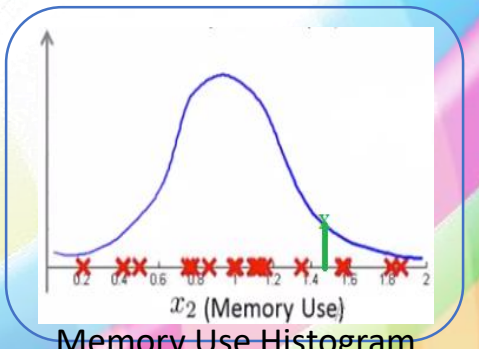
Following is how anomaly detection algo works:



CPU Load vs Memory Use



CPU Load Histogram



Memory Use Histogram

From the plot “CPU Load vs Memory Use”, it is clear that the value of green cross mark is 0.4 on “CPU Load” axis and 1.5 on “Memory Use” axis.

Anomaly detection algorithm will see the probability of 0.4 on “CPU Load Histogram” plot and 1.5 on “Memory Use Histogram”. Since both these probabilities are reasonably high, hence their product will give a value which will be greater than ε, resulting into “normal” or “non-anomalous” prediction. However, it is clear from the plot, as well as logically that the point is an anomaly as memory usage should not be high (i.e., 1.5) when CPU load is quite less (i.e., 0.4).

To fix this, we will use something called as “Multivariate Gaussian Distribution” in anomaly detection algorithm.

Multivariate Gaussian Distribution

Multivariate Gaussian Distribution

- Multivariate Gaussian Distribution is also known as Multivariate Normal Distribution.
- Multivariate Gaussian Distribution does not model $p(x_1)$, $p(x_2)$, …., $p(x_n)$, separately. Instead, it models p(x) in one go.
- Following are the parameters of Multivariate Gaussian Distribution: μ, Σ. Where μ is mean and Σ is covariance matrix.
- Following is the formula of Multivariate Gaussian Matrix:

$$p(x;\ ,\ ) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}\ exp\left(-\frac{1}{2}\ (x-\ )^T\Sigma^{-1}(x-\ )\right)$$

Where, $|\Sigma|$ is the determinant of covariance matrix Σ.

Multivariate Gaussian Distribution

Anomaly Detection Using The Multivariate Gaussian Detection

- Anomaly detection with the Multivariate Gaussian Distribution:

  Suppose given dataset is: $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

  1. Calculate: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$

     Where, μ is mean and Σ is covariance matrix.

  2. For a new example, x, compute:

$$p(x) = \frac{1}{2\pi^{n/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

     Flag as anomaly if $p(x) < \varepsilon$

Original Model	Multivariate Gaussian Model
$p(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\,\sigma_j} \exp\left(-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}\right)$	$p(x) = \frac{1}{2\pi^{n/2}\lvert\Sigma\rvert^{1/2}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$
1. Manually create features to capture anomalies where $x_1, x_2, .., x_n$ take unusual combinations of values.	1. Automatically captures correlation between features.
2. Computationally cheaper.	2. Computationally more expensive.
3. Okay even if m (training set size) is small.	3. Must have m > n or else Σ is non-invertible.

Predicting Movie Ratings

Problem Formulation

- Following is our data:

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Romance (x_1)	Action (x_2)
Love At Lat (1)	5	5	0	0	0.9	0
Romance Forever (2)	5	?	?	0	1.0	0.01
Cute Puppies of Love (3)	?	4	0	?	0.99	0
Nonstop Car Chases (4)	0	0	5	4	0.1	1.0
Swords vs Karate (5)	0	0	5	?	0	0.9

- In the above dataset, number of users (n_u) is 4 (Alice, Bob, Carol and Dave) and number of movies (n_m) is 5.
- We are using 0 to 5 rating system, although many websites use 1 to 5 rating system.
- Now, our task is to predict the missing values, i.e., "?", i.e:
  1. To predict the response of these users OR
  2. To predict the recommendation on the scale of 0 to 5 for these users.
- Now, we have two features, romance (denoted by x_1) and action (denoted by x_2). The degree of romance and action of these movies is given in the above table.

Predicting Movie Ratings

Content Based Recommendation

- There are two major recommendation techniques:
  1. Content Based Recommendation
  2. Collaborative Filtering
- Following are the feature vectors of each of five movies:

$$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix} \quad x^{(2)} = \begin{bmatrix} 1 \\ 1.0 \\ 0.01 \end{bmatrix} \quad x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \quad x^{(4)} = \begin{bmatrix} 1 \\ 0.1 \\ 1.0 \end{bmatrix} \quad x^{(5)} = \begin{bmatrix} 1 \\ 0 \\ 0.9 \end{bmatrix}$$

  We have added an extra feature for all movies, i.e., "1".
- r(i, j): 1 if user j has rated movie i (0 otherwise) and y(i, j) is rating by user j for movie i (if defined).
- Based on early ratings given by each user, we will have a feature vector for each user, represented by $\theta^{(i)}$. Thus, the rating for movie i by user j is given by: $y(i, j) = (\theta^{(j)})^T x^{(i)}$
- Now, suppose we want to predict the rating or recommendation for movie "Cute Puppies of Love" for user Alice (1). From earlier choices of Alice, some algorithm has made the following feature for Alice (1):

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$$

- Thus, the rating prediction or recommendation for movie 3 by user 1 would be:

$$y(3,1) = (\theta^{(1)})^T x^{(3)} = 5\, x\, 0.99 = 4.95$$

Predicting Movie Ratings

Content Based Recommendation

- Let us say:
  $m^{(j)}$: Number of movies rated by user j.
  To learn $\theta^{(j)}$:

Skipped

Collaborative Filtering

Collaborative Filtering

- In the previous recommendation technique, it was required that somebody should watch each movie and define how romantic or action it has, i.e., features x_1 and x_2. Since, it is quite time consuming, subjective and often requires lot more features, hence another recommendation technique comes into picture: Collaborative Filtering.
- Suppose, we have following data:

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Romance (x_1)	Action (x_2)
Love At Lat (1)	5	5	0	0	?	?
Romance Forever (2)	5	?	?	0	?	?
Cute Puppies of Love (3)	?	4	0	?	?	?
Nonstop Car Chases (4)	0	0	5	4	?	?
Swords vs Karate (5)	0	0	5	?	?	?

- Now, we don't know the values of our movie features, i.e., x_1 and x_2. However, we got the values of user parameters:

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \quad \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \quad \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} \quad \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

Collaborative Filtering

Collaborative Filtering

- We have to figure out $x^{(1)}$ in such a manner so that following get satisfied:

  $(\theta^{(1)})^T x^{(1)} \approx 5$ $(\theta^{(2)})^T x^{(1)} \approx 5$ $(\theta^{(3)})^T x^{(1)} \approx 0$ $(\theta^{(4)})^T x^{(1)} \approx 0$

  On solving above equations, we will get:

$$x^{(1)} = \begin{bmatrix} 1 \\ 1.0 \\ 0 \end{bmatrix}$$

  Similarly, other movie features can also be find.

Skipped

Collaborative Filtering

Collaborative Filtering Algorithm

Skipped

Low Rank Matrix Factorization

Vectorization: Low Rank Matrix Factorization

Skipped

Low Rank Matrix Factorization

Implementational Detail: Mean Normalization

Skipped

Gradient Descent With Large Datasets

Learning With Large Datasets

- Following is the update step of Gradient Descent: $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h(x^i) - y^i\right) . x_j^i$

  Now, for a large dataset, m would be very large, say m = 100,000,000 (i.e., 100 million). Then, calculating the above computation for such a huge dataset will be very time and computation expensive.

  One trick to deal with this kind of situation is just to train model on 1,000 instances instead of 100 million. It is possible that training on only 1,000 instances might do the job as great as 100 million instances.

- Suppose you are facing a supervised learning problem and have a very large dataset (m = 100,000,000). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say, m = 1,000)?

  Ans: Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

  The first figure is showing this situation, i.e., the situation of high variance when a large dataset would be helpful for better results.

  Whereas, the second figure is showing the situation of high bias, i.e., the large dataset will not help in achieving better results.

  However, if you have got the situation like in fig. 2, then try adding extra features or more hidden units to get situation like showing in figure 1.

Figure 1

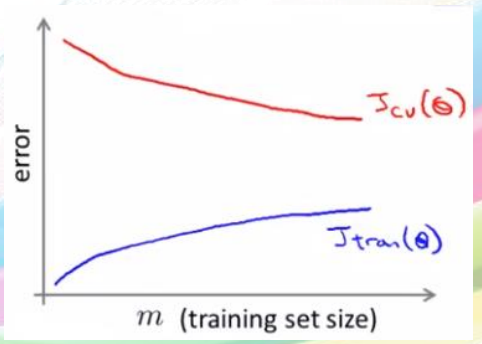
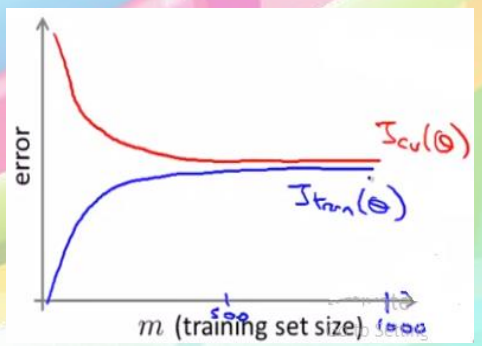


Figure 2



Gradient Descent With Large Datasets

Stochastic Gradient Descent

- For many machine learning algorithms, like Linear Regression, Logistic Regression, Neural Networks, etc., we have a cost function & an optimization function, like Gradient Descent, that we use to minimize the cost. When we have very large training set, these optimization functions, like Gradient Descent, becomes computationally expensive.
- A modification of Gradient Descent, called as Stochastic Gradient Descent, which allow us to scale these machine learning algorithms to much larger training sets.
- The Gradient Descent that uses the entire training set as a single batch, is called as Batch Gradient Descent.
- For a large training set, say m = 300,000,000, Batch Gradient Descent has to read this entire 300,000,000 instances and keep them in memory for computations. This is how one step of Gradient Descent will complete.
- In contrast to Batch Gradient Descent, Stochastic Gradient Descent doesn't need to look at entire data in one go. This algorithm looks at a single training instance in an iteration.

- Following are the steps of Stochastic Gradient Descent:
  1. Randomly shuffle dataset.
  2. Repeat:

```
For i = 1, 2, …, m {
        For j = 0, 1, …, n {
                θ_j := θ_j − α(h_θ(x^(i)) − y^(i)). x_j^(i)
        }
}
```

- The advantage of using Stochastic Gradient Descent is that instead of waiting to read and make a tiny update towards global minimum after reading a huge training set (like Batch Gradient Descent), it reads one training instance and keep updating parameters towards global minimum.

Gradient Descent With Large Datasets

Mini Batch Gradient Descent

- Mini Batch Gradient Descent is another variation of Gradient Descent.
- Earlier, we saw that Stochastic Gradient Descent is faster than Batch Gradient Descent.
- Sometimes, Mini Batch Gradient Descent works faster than even Stochastic Gradient Descent.
- Following is a summary of all these variants of Gradient Descent algorithm:
  - Batch Gradient Descent: It uses all m examples in each iteration.
  - Stochastic Gradient Descent: It uses only 1 example in each iteration.
  - Mini Batch Gradient Descent: It uses b examples in each iteration.
- Typical value of mini batch, b, could be 10, or 2 to 100.
- It uses b number of examples for Gradient Descent update.
- Compared to Batch Gradient Descent, Mini Batch Gradient Descent allows faster update.
- Mini Batch Gradient Descent can outperform Stochastic Gradient Descent if it is implemented properly by using vectorization as it parallelizes the computations.
- One tiny disadvantage of Mini Batch Gradient Descent is the presence of one more variable, b.

Gradient Descent With Large Datasets

Stochastic Gradient Descent Convergence

- We now know Stochastic Gradient Descent algorithm. Now, we will see how to figure out if the convergence is okay and how to select the learning rate.
- Earlier for Batch Gradient Descent we were checking convergence by plotting $J_{train}(\theta)$ as a function of the number of iterations of gradient descent and we were assuring that the cost function is decreasing at every iteration. This was possible to do if the size of training set is small. However, in case of a huge dataset, this will take a lot of time.
- Whereas, in case of Stochastic Gradient Descent, cost can be calculated by:

  $$cost\left(\theta, \left(x^{(i)}, y^{(i)}\right)\right) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

  During learning, computer cost as given by above equation before updating θ using $(x^{(i)}, y^{(i)})$.
  Every 1,000 iteration (say), plot cost averaged over last 1,000 examples processed by algorithm.
  By looking at these plots, one can figure out if Stochastic Gradient Descent is converging or not.
- Learning Rate (α) is typically held constant. It can be slowly decreased over time if we want θ to converge. For example, $\alpha = \frac{constant\_1}{iterationNumber + constant\_2}$. However, people generally do not try this technique because it further increases the headache of tuning two more parameters, constant_1 and constant_2. This formula to set learning rate (α) makes sense because it decreases learning rate after each iteration as number of iteration increases as iterationNumber is in denominator.

Advanced Topics

Online Learning

- Online Learning is useful when we want our machine learning algorithm to learn from a continuous stream of data.
- Following is an example:

  A shipping service website where user comes and enters specific origin and destination, website offers some price for that shipping and then user choose to ship (y = 1) or not to ship (y = 0).

  Feature x captures properties of user, such as origin, destination and price offered. We want to learn p(y=1| x; θ) to optimize price. Here, y is our output (1, if user opts the shipping service at the offered price and 0 if user doesn't), x is input feature that comprises source, destination and offered price, and θ is the parameters of our learning algorithm (say Logistic Regression).

  Following is out algorithm:

  Repeat forever {

  Get (x, y) corresponding to user

  Update θ using (x, y):

  $$\theta_j := \theta_j - \alpha\left(h_\theta(x) - y\right).x_j$$

  }

  In online learning, algorithm learns from each example and then discard that example, i.e., never learns it from that again. Due to this notion, there is no need to use indexes "I" around (x, y).

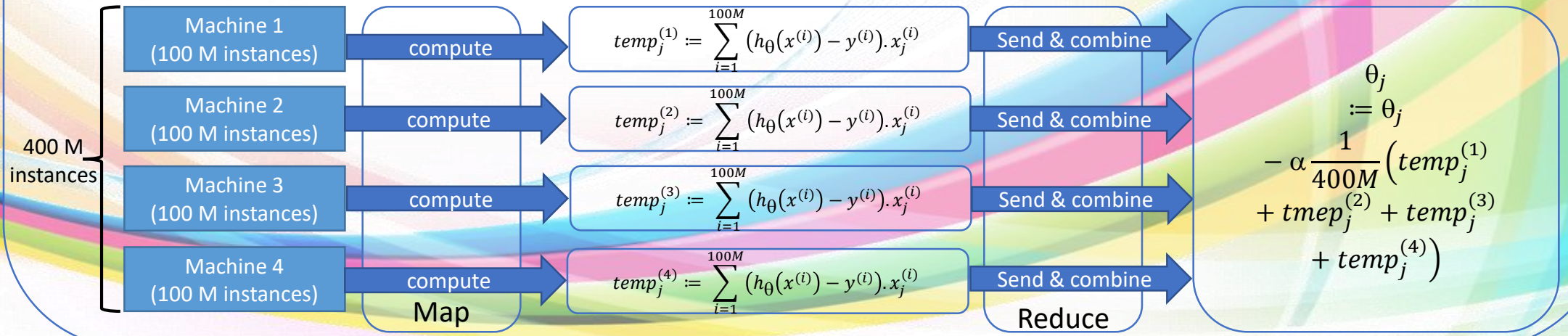
  One advantage of online learning is that it can quickly adopt to changing user preferences. For instance, in a growing economy, users may get ready to pay high prices.

Advanced Topics

Map Reduce And Data Parallelism

- All previous machine learning algorithms that we have seen so far can learn only on one machine.
- Sometimes, some machine learning algorithms are just too big to run on a single machine or we have a huge dataset spread over several machines. To address this problem, we will discuss a new approach to scale machine learning algorithms, called as Map Reduce.
- Following is an example of Map Reduce: assume we have 400 Million instances spread over 4 machines in equal parts of 100 M each and following is our cost function: $\theta_j = \theta_j - \alpha \frac{1}{400M} \sum_{i=1}^{400M} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).x_j^{(i)}$

  Since the summation part of above cost function is computationally very expensive, hence following is how Map Reduce will work and make it drastically cheap:



Advanced Topics

Map Reduce And Data parallelism

- Map Reduce can also be utilised on machines having multiple cores. So on such machines, a huge training set can be split into c (i.e., number of cores) parts and send on each of them for further processing.
- Following figure is depicting the same:
- Suppose you apply the map-reduce method to train a neural network on ten machines. In each iteration, what will each of the machines do?

  Ans. Compute forward propagation and backward propagation on 1/10 of the data to compute the derivative with respect to that 1/10 of the data.

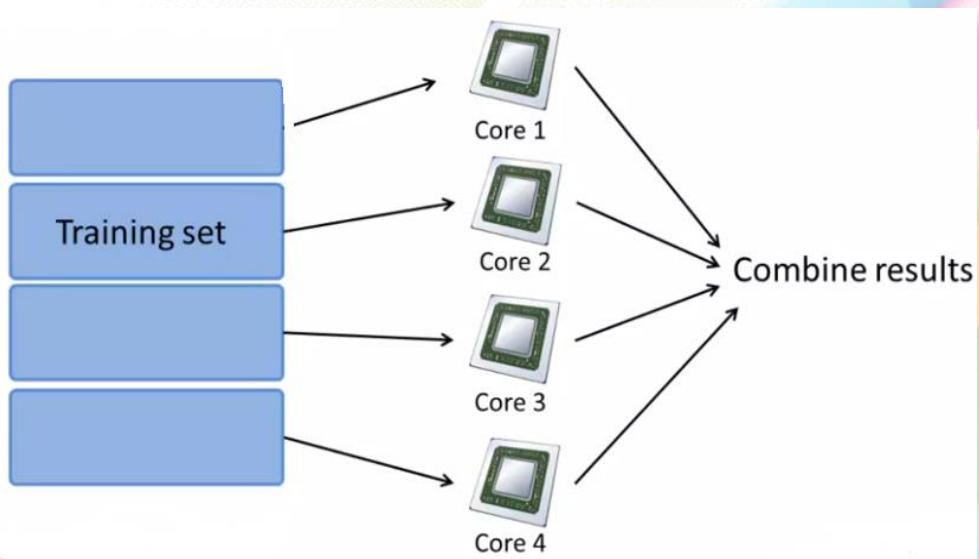


Photo OCR

Problem Description And Pipeline

- Photo OCR stands for Photo Optical Character Recognition.
- If a computer can read characters inside a photo then that would be helpful for us to quickly get a picture having a label or brand name in it instead of looking several dozens of picture.
- Doing OCR on scanned documents is significantly easier than OCR on photos.
- OCR can also help a blind person by reading and telling him the posters and instructions written on a street.
- Researchers are also working on making use of OCR in car navigation system.
- Following is the pipeline of OCR:

1. Text Detection
2. Character Segmentation
3. Character Classification

- Some advanced OCR systems also does spelling correction, however here we will not consider that part.

Photo OCR

Sliding Windows

- The most complicated task or module of Photo OCR is Text Detection.
- Since in this task we have to mark text of varying length, width (i.e., aspect ratio) and color (as shown in this figure), hence it becomes hard.

Skipped

Photo OCR

Getting Lots Of Data And Artificial Data

- There is an idea of Artificial Data Synthesis that doesn't apply on all data but if it applies on your data then it can help you to have lot of data.
- There are two ways of synthesising Artificial Data:
  1. Creating data from scratch
  2. Amplifying already existing and labelled data
- For Photo OCR, one can artificially generate data by utilising the font libraries available in our computers.
- These generated images can be distorted by adding noise or blurring.
- Suppose you are training a Linear Regression model with m examples and you have duplicated every example by making two identical copies of it. That is, previously you had one example $(x^{(i)}, y^{(i)})$, now you have two copies of it. So now you have 2m examples. Is this likely to help?

  Ans. No. In fact, you will end up with the same parameters as before you duplicated the data.
- Before increasing data either by collecting or generating, you should assure:
  - To have a low bias classifier before expending the effort (plot learning curves). Example, keep increasing the number of features / number of hidden units in neural network until you have a low bias classifier.
  - How much work would it be to get 10x data as much as we currently have?
    - Artificial Data Synthesis
    - Collect / label it yourself
    - Crowd Source – Amazon Mechanical Turk

Photo OCR

Ceiling Analysis: What Part Of The Pipeline To Work On Next

- What part of the pipeline should you spend the most time trying to improve?

Skipped

Week

Template

- Template

Thank You